Machine Learning for Software Analysis (MLSA)

University of Florence -- IMT School for Advanced Studies Lucca

Fabio Pinelli

fabio.pinelli@imtlucca.it (mailto:fabio.pinelli@imtlucca.it)
IMT School for Advanced Studies Lucca

2025/2026
November, 11 2025

```
In [ ]: ############################################################
        !pip install gensim==4.3.3
        # The library has been archived and won't be used anymore
        # # !pip install allennlp==0.9.0
        !pip install flair==0.13.1
        !pip install torchvision==0.18.1
        # # HuggingFace

        !pip uninstall -y transformers peft
        !pip install transformers==4.38.0
        !pip install datasets==2.18.0
        !pip install peft==0.8.2
        !pip install accelerate==0.30.0
        ############################################################
```

```
Requirement already satisfied: gensim==4.3.3 in /usr/local/lib/python3.12/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim==4.
3.3) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim==
4.3.3) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim==4.3.
3) (7.5.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart-open>=1.8.1->gensi
m==4.3.3) (2.0.1)
Requirement already satisfied: flair==0.13.1 in /usr/local/lib/python3.12/dist-packages (0.13.1)
Requirement already satisfied: boto3>=1.20.27 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1)
(1.41.3)
Requirement already satisfied: bpemb>=0.3.2 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1)
(0.3.6)
Requirement already satisfied: conllu>=4.0 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1) (6.
0.0)
Requirement already satisfied: deprecated>=1.2.13 in /usr/local/lib/python3.12/dist-packages (from flair==0.1
3.1) (1.3.1)
Requirement already satisfied: ftfy>=6.1.0 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1) (6.
3.1)
Requirement already satisfied: gdown>=4.4.0 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1)
(5.2.0)
Requirement already satisfied: gensim>=4.2.0 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1)
(4.3.3)
Requirement already satisfied: huggingface-hub>=0.10.0 in /usr/local/lib/python3.12/dist-packages (from flair=
=0.13.1) (0.36.0)
Requirement already satisfied: janome>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1)
(0.5.0)
Requirement already satisfied: langdetect>=1.0.9 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.
1) (1.0.9)
Requirement already satisfied: lxml>=4.8.0 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1) (6.
0.2)
Requirement already satisfied: matplotlib>=2.2.3 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.
1) (3.10.0)
Requirement already satisfied: more-itertools>=8.13.0 in /usr/local/lib/python3.12/dist-packages (from flair==
0.13.1) (10.8.0)
Requirement already satisfied: mpld3>=0.3 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1) (0.
5.12)
Requirement already satisfied: pptree>=3.1 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1) (3.
1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from flair==
0.13.1) (2.9.0.post0)
Requirement already satisfied: pytorch-revgrad>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from flair==
0.13.1) (0.2.0)
Requirement already satisfied: regex>=2022.1.18 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.
1) (2025.11.3)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.12/dist-packages (from flair==0.1
3.1) (1.6.1)
Requirement already satisfied: segtok>=1.5.11 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1)
(1.5.11)
Requirement already satisfied: sqlitedict>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.
1) (2.1.0)
Requirement already satisfied: tabulate>=0.8.10 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.
1) (0.9.0)
Requirement already satisfied: torch!=1.8,>=1.5.0 in /usr/local/lib/python3.12/dist-packages (from flair==0.1
3.1) (2.3.1)
Requirement already satisfied: tqdm>=4.63.0 in /usr/local/lib/python3.12/dist-packages (from flair==0.13.1)
(4.67.1)
Requirement already satisfied: transformer-smaller-training-vocab>=0.2.3 in /usr/local/lib/python3.12/dist-pac
kages (from flair==0.13.1) (0.4.2)
Requirement already satisfied: transformers<5.0.0,>=4.18.0 in /usr/local/lib/python3.12/dist-packages (from tr
ansformers[sentencepiece]<5.0.0,>=4.18.0->flair==0.13.1) (4.38.0)
Requirement already satisfied: urllib3<2.0.0,>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from flair==
0.13.1) (1.26.20)
Requirement already satisfied: wikipedia-api>=0.5.7 in /usr/local/lib/python3.12/dist-packages (from flair==0.
13.1) (0.8.1)
Requirement already satisfied: semver<4.0.0,>=3.0.0 in /usr/local/lib/python3.12/dist-packages (from flair==0.
13.1) (3.0.4)
Requirement already satisfied: botocore<1.42.0,>=1.41.3 in /usr/local/lib/python3.12/dist-packages (from boto3
>=1.20.27->flair==0.13.1) (1.41.3)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/local/lib/python3.12/dist-packages (from boto3>=
1.20.27->flair==0.13.1) (1.0.1)
Requirement already satisfied: s3transfer<0.16.0,>=0.15.0 in /usr/local/lib/python3.12/dist-packages (from bot
o3>=1.20.27->flair==0.13.1) (0.15.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from bpemb>=0.3.2->flair==0.1
3.1) (1.26.4)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from bpemb>=0.3.2->flair==
0.13.1) (2.32.4)
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.12/dist-packages (from bpemb>=0.3.2->fl
air==0.13.1) (0.2.1)
Requirement already satisfied: wrapt<3,>=1.10 in /usr/local/lib/python3.12/dist-packages (from deprecated>=1.
2.13->flair==0.13.1) (2.0.1)
```

```
Requirement already satisfied: wcwidth in /usr/local/lib/python3.12/dist-packages (from ftfy>=6.1.0->flair==0.
13.1) (0.2.14)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dist-packages (from gdown>=4.4.0->f
lair==0.13.1) (4.13.5)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from gdown>=4.4.0->flair==
0.13.1) (3.20.0)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim>=
4.2.0->flair==0.13.1) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim>=4.2.
0->flair==0.13.1) (7.5.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-h
ub>=0.10.0->flair==0.13.1) (2024.2.0)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.12/dist-packages (from huggingface-hu
b>=0.10.0->flair==0.13.1) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=
0.10.0->flair==0.13.1) (6.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from hug
gingface-hub>=0.10.0->flair==0.13.1) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingfa
ce-hub>=0.10.0->flair==0.13.1) (1.2.0)
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-packages (from langdetect>=1.0.9->flair==
0.13.1) (1.17.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=
2.2.3->flair==0.13.1) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=2.2.3
->flair==0.13.1) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=
2.2.3->flair==0.13.1) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=
2.2.3->flair==0.13.1) (1.4.9)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=2.2.3->f
lair==0.13.1) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=
2.2.3->flair==0.13.1) (3.2.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from mpld3>=0.3->flair==0.1
3.1) (3.1.6)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=1.
0.2->flair==0.13.1) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-le
arn>=1.0.2->flair==0.13.1) (3.6.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.12/dist-packages (from torch!=1.8,>=1.5.0->flai
r==0.13.1) (1.14.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch!=1.8,>=1.5.0->f
lair==0.13.1) (3.5)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (fr
om torch!=1.8,>=1.5.0->flair==0.13.1) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages
(from torch!=1.8,>=1.5.0->flair==0.13.1) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (fr
om torch!=1.8,>=1.5.0->flair==0.13.1) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.12/dist-packages (from to
rch!=1.8,>=1.5.0->flair==0.13.1) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.12/dist-packages (from t
orch!=1.8,>=1.5.0->flair==0.13.1) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.12/dist-packages (from t
orch!=1.8,>=1.5.0->flair==0.13.1) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.12/dist-packages (from
torch!=1.8,>=1.5.0->flair==0.13.1) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.12/dist-packages (fr
om torch!=1.8,>=1.5.0->flair==0.13.1) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.12/dist-packages (fr
om torch!=1.8,>=1.5.0->flair==0.13.1) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.20.5 in /usr/local/lib/python3.12/dist-packages (from torc
h!=1.8,>=1.5.0->flair==0.13.1) (2.20.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (from tor
ch!=1.8,>=1.5.0->flair==0.13.1) (12.1.105)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.12/dist-packages (from nvidia-c
usolver-cu12==11.4.5.107->torch!=1.8,>=1.5.0->flair==0.13.1) (12.6.85)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python3.12/dist-packages (from transfo
rmers<5.0.0,>=4.18.0->transformers[sentencepiece]<5.0.0,>=4.18.0->flair==0.13.1) (0.15.2)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.12/dist-packages (from transformer
s<5.0.0,>=4.18.0->transformers[sentencepiece]<5.0.0,>=4.18.0->flair==0.13.1) (0.7.0)
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-packages (from transformers[sentence
piece]<5.0.0,>=4.18.0->flair==0.13.1) (5.29.5)
Requirement already satisfied: accelerate>=0.21.0 in /usr/local/lib/python3.12/dist-packages (from transformer
s[sentencepiece,torch]<5.0,>=4.1->transformer-smaller-training-vocab>=0.2.3->flair==0.13.1) (0.30.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4->
gdown>=4.4.0->flair==0.13.1) (2.8)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->mpld3>
=0.3->flair==0.13.1) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reque
sts->bpemb>=0.3.2->flair==0.13.1) (3.4.4)
```

```
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->bpemb>=
0.3.2->flair==0.13.1) (3.11)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->b
pemb>=0.3.2->flair==0.13.1) (2025.11.12)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.12/dist-packages (from request
s[socks]->gdown>=4.4.0->flair==0.13.1) (1.7.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy->torc
h!=1.8,>=1.5.0->flair==0.13.1) (1.3.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from accelerate>=0.21.0->tra
nsformers[sentencepiece,torch]<5.0,>=4.1->transformer-smaller-training-vocab>=0.2.3->flair==0.13.1) (5.9.5)
Requirement already satisfied: torchvision==0.18.1 in /usr/local/lib/python3.12/dist-packages (0.18.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from torchvision==0.18.1) (1.
26.4)
Requirement already satisfied: torch==2.3.1 in /usr/local/lib/python3.12/dist-packages (from torchvision==0.1
8.1) (2.3.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from torchvis
ion==0.18.1) (11.3.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch==2.3.1->torchvi
sion==0.18.1) (3.20.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.12/dist-packages (from torch
==2.3.1->torchvision==0.18.1) (4.15.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.12/dist-packages (from torch==2.3.1->torchvisio
n==0.18.1) (1.14.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch==2.3.1->torchvi
sion==0.18.1) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch==2.3.1->torchvisi
on==0.18.1) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from torch==2.3.1->torchvisi
on==0.18.1) (2024.2.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (fr
om torch==2.3.1->torchvision==0.18.1) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages
(from torch==2.3.1->torchvision==0.18.1) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (fr
om torch==2.3.1->torchvision==0.18.1) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.12/dist-packages (from to
rch==2.3.1->torchvision==0.18.1) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.12/dist-packages (from t
orch==2.3.1->torchvision==0.18.1) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.12/dist-packages (from t
orch==2.3.1->torchvision==0.18.1) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.12/dist-packages (from
torch==2.3.1->torchvision==0.18.1) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.12/dist-packages (fr
om torch==2.3.1->torchvision==0.18.1) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.12/dist-packages (fr
om torch==2.3.1->torchvision==0.18.1) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.20.5 in /usr/local/lib/python3.12/dist-packages (from torch
==2.3.1->torchvision==0.18.1) (2.20.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (from tor
ch==2.3.1->torchvision==0.18.1) (12.1.105)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.12/dist-packages (from nvidia-c
usolver-cu12==11.4.5.107->torch==2.3.1->torchvision==0.18.1) (12.6.85)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch=
=2.3.1->torchvision==0.18.1) (3.0.3)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy->torc
h==2.3.1->torchvision==0.18.1) (1.3.0)
Found existing installation: transformers 4.38.0
Uninstalling transformers-4.38.0:
  Successfully uninstalled transformers-4.38.0
WARNING: Skipping peft as it is not installed.
Collecting transformers==4.38.0
  Using cached transformers-4.38.0-py3-none-any.whl.metadata (131 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from transformers==4.38.0)
(3.20.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in /usr/local/lib/python3.12/dist-packages (from t
ransformers==4.38.0) (0.36.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from transformers==4.3
8.0) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers==
4.38.0) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers==4.3
8.0) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers
==4.38.0) (2025.11.3)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from transformers==4.38.0)
(2.32.4)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python3.12/dist-packages (from transfo
rmers==4.38.0) (0.15.2)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.12/dist-packages (from transformer
s==4.38.0) (0.7.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (from transformers==4.38.
0) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-h
ub<1.0,>=0.19.3->transformers==4.38.0) (2024.2.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from hug
gingface-hub<1.0,>=0.19.3->transformers==4.38.0) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingfa
ce-hub<1.0,>=0.19.3->transformers==4.38.0) (1.2.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reque
sts->transformers==4.38.0) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->transfo
rmers==4.38.0) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->t
ransformers==4.38.0) (1.26.20)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->t
ransformers==4.38.0) (2025.11.12)
Using cached transformers-4.38.0-py3-none-any.whl (8.5 MB)
Installing collected packages: transformers
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. Thi
s behaviour is the source of the following dependency conflicts.
sentence-transformers 5.1.2 requires transformers<5.0.0,>=4.41.0, but you have transformers 4.38.0 which is in
compatible.
Successfully installed transformers-4.38.0
Requirement already satisfied: datasets==2.18.0 in /usr/local/lib/python3.12/dist-packages (2.18.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.0) (3.
20.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.0)
(1.26.4)
Requirement already satisfied: pyarrow>=12.0.0 in /usr/local/lib/python3.12/dist-packages (from datasets==2.1
8.0) (18.1.0)
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.
0) (0.7)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from datasets==
2.18.0) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.0) (2.2.
2)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.12/dist-packages (from datasets==2.1
8.0) (2.32.4)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.0)
(4.67.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.0) (3.6.
0)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.0)
(0.70.16)
Requirement already satisfied: fsspec<=2024.2.0,>=2023.1.0 in /usr/local/lib/python3.12/dist-packages (from fs
spec[http]<=2024.2.0,>=2023.1.0->datasets==2.18.0) (2024.2.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.0) (3.1
3.2)
Requirement already satisfied: huggingface-hub>=0.19.4 in /usr/local/lib/python3.12/dist-packages (from datase
ts==2.18.0) (0.36.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.0) (2
5.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from datasets==2.18.0)
(6.0.3)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohtt
p->datasets==2.18.0) (2.6.1)
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->data
sets==2.18.0) (1.4.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->dataset
s==2.18.0) (25.4.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp->dat
asets==2.18.0) (1.8.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp->d
atasets==2.18.0) (6.7.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->data
sets==2.18.0) (0.4.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp->dat
asets==2.18.0) (1.22.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from hug
gingface-hub>=0.19.4->datasets==2.18.0) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingfa
ce-hub>=0.19.4->datasets==2.18.0) (1.2.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reque
sts>=2.19.0->datasets==2.18.0) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0-
>datasets==2.18.0) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=
2.19.0->datasets==2.18.0) (1.26.20)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=
2.19.0->datasets==2.18.0) (2025.11.12)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas-
>datasets==2.18.0) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets=
=2.18.0) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas->dataset
s==2.18.0) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.
2->pandas->datasets==2.18.0) (1.17.0)
Collecting peft==0.8.2
  Using cached peft-0.8.2-py3-none-any.whl.metadata (25 kB)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from peft==0.8.2) (1.2
6.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from peft==0.8.2)
(25.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from peft==0.8.2) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages (from peft==0.8.2) (6.0.3)
Requirement already satisfied: torch>=1.13.0 in /usr/local/lib/python3.12/dist-packages (from peft==0.8.2) (2.
3.1)
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (from peft==0.8.2) (4.3
8.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from peft==0.8.2) (4.67.1)
Requirement already satisfied: accelerate>=0.21.0 in /usr/local/lib/python3.12/dist-packages (from peft==0.8.
2) (0.30.0)
Requirement already satisfied: safetensors in /usr/local/lib/python3.12/dist-packages (from peft==0.8.2) (0.7.
0)
Requirement already satisfied: huggingface-hub>=0.17.0 in /usr/local/lib/python3.12/dist-packages (from peft==
0.8.2) (0.36.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.1
7.0->peft==0.8.2) (3.20.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-h
ub>=0.17.0->peft==0.8.2) (2024.2.0)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.1
7.0->peft==0.8.2) (2.32.4)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from hug
gingface-hub>=0.17.0->peft==0.8.2) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingfa
ce-hub>=0.17.0->peft==0.8.2) (1.2.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.12/dist-packages (from torch>=1.13.0->peft==0.
8.2) (1.14.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch>=1.13.0->peft==
0.8.2) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.13.0->peft==0.
8.2) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (fr
om torch>=1.13.0->peft==0.8.2) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages
(from torch>=1.13.0->peft==0.8.2) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (fr
om torch>=1.13.0->peft==0.8.2) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.12/dist-packages (from to
rch>=1.13.0->peft==0.8.2) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.12/dist-packages (from t
orch>=1.13.0->peft==0.8.2) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.12/dist-packages (from t
orch>=1.13.0->peft==0.8.2) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.12/dist-packages (from
torch>=1.13.0->peft==0.8.2) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.12/dist-packages (fr
om torch>=1.13.0->peft==0.8.2) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.12/dist-packages (fr
om torch>=1.13.0->peft==0.8.2) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.20.5 in /usr/local/lib/python3.12/dist-packages (from torch
>=1.13.0->peft==0.8.2) (2.20.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (from tor
ch>=1.13.0->peft==0.8.2) (12.1.105)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.12/dist-packages (from nvidia-c
usolver-cu12==11.4.5.107->torch>=1.13.0->peft==0.8.2) (12.6.85)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers
->peft==0.8.2) (2025.11.3)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python3.12/dist-packages (from transfo
rmers->peft==0.8.2) (0.15.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>
=1.13.0->peft==0.8.2) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reque
sts->huggingface-hub>=0.17.0->peft==0.8.2) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->hugging
face-hub>=0.17.0->peft==0.8.2) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->h
uggingface-hub>=0.17.0->peft==0.8.2) (1.26.20)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->h
uggingface-hub>=0.17.0->peft==0.8.2) (2025.11.12)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy->torc
h>=1.13.0->peft==0.8.2) (1.3.0)
Using cached peft-0.8.2-py3-none-any.whl (183 kB)
Installing collected packages: peft
```

```
Successfully installed peft-0.8.2
Requirement already satisfied: accelerate==0.30.0 in /usr/local/lib/python3.12/dist-packages (0.30.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from accelerate==0.30.
0) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from accelerate==0.
30.0) (25.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from accelerate==0.30.0) (5.
9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages (from accelerate==0.30.0) (6.
0.3)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.12/dist-packages (from accelerate==0.3
0.0) (2.3.1)
Requirement already satisfied: huggingface-hub in /usr/local/lib/python3.12/dist-packages (from accelerate==0.
30.0) (0.36.0)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.12/dist-packages (from accelerate=
=0.30.0) (0.7.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch>=1.10.0->accele
rate==0.30.0) (3.20.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.12/dist-packages (from torch
>=1.10.0->accelerate==0.30.0) (4.15.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.12/dist-packages (from torch>=1.10.0->accelerat
e==0.30.0) (1.14.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch>=1.10.0->accele
rate==0.30.0) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.10.0->accelera
te==0.30.0) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from torch>=1.10.0->accelera
te==0.30.0) (2024.2.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (fr
om torch>=1.10.0->accelerate==0.30.0) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages
(from torch>=1.10.0->accelerate==0.30.0) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (fr
om torch>=1.10.0->accelerate==0.30.0) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.12/dist-packages (from to
rch>=1.10.0->accelerate==0.30.0) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.12/dist-packages (from t
orch>=1.10.0->accelerate==0.30.0) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.12/dist-packages (from t
orch>=1.10.0->accelerate==0.30.0) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.12/dist-packages (from
torch>=1.10.0->accelerate==0.30.0) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.12/dist-packages (fr
om torch>=1.10.0->accelerate==0.30.0) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.12/dist-packages (fr
om torch>=1.10.0->accelerate==0.30.0) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.20.5 in /usr/local/lib/python3.12/dist-packages (from torch
>=1.10.0->accelerate==0.30.0) (2.20.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.12/dist-packages (from tor
ch>=1.10.0->accelerate==0.30.0) (12.1.105)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.12/dist-packages (from nvidia-c
usolver-cu12==11.4.5.107->torch>=1.10.0->accelerate==0.30.0) (12.6.85)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from huggingface-hub->acce
lerate==0.30.0) (2.32.4)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub->
accelerate==0.30.0) (4.67.1)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingfa
ce-hub->accelerate==0.30.0) (1.2.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>
=1.10.0->accelerate==0.30.0) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reque
sts->huggingface-hub->accelerate==0.30.0) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->hugging
face-hub->accelerate==0.30.0) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->h
uggingface-hub->accelerate==0.30.0) (1.26.20)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->h
uggingface-hub->accelerate==0.30.0) (2025.11.12)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy->torc
h>=1.10.0->accelerate==0.30.0) (1.3.0)
```

```
In [ ]: try:
            import google.colab
            import requests
            url = 'https://raw.githubusercontent.com/dvgodoy/PyTorchStepByStep/master/config.py'
            r = requests.get(url, allow_redirects=True)
            open('config.py', 'wb').write(r.content)
        except ModuleNotFoundError:
            pass

        from config import *
        config_chapter11()
        # This is needed to render the plots in this chapter
        from plots.chapter11 import *
```

```
Downloading files from GitHub repo to Colab...
Finished!
```

```
In [ ]: import os
        import json
        import errno
        import requests
        import numpy as np
        from copy import deepcopy
        from operator import itemgetter

        import torch
        import torch.optim as optim
        import torch.nn as nn
        import torch.nn.functional as F
        from torch.utils.data import DataLoader, TensorDataset, Dataset

        from data_generation.nlp import ALICE_URL, WIZARD_URL, download_text
        from stepbystep.v4 import StepByStep
        # These are the classes we built in previous class 10
        from seq2seq import *

        import nltk
        from nltk.tokenize import sent_tokenize
        nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
Out[ ]: True
```

```
In [ ]: import gensim
        from gensim import corpora, downloader
        from gensim.parsing.preprocessing import *
        from gensim.utils import simple_preprocess
        from gensim.models import Word2Vec
```

```
In [ ]: from flair.data import Sentence
        #from flair.embeddings import ELMoEmbeddings, WordEmbeddings, \
        #    TransformerWordEmbeddings, TransformerDocumentEmbeddings
        from flair.embeddings import WordEmbeddings, \
            TransformerWordEmbeddings, TransformerDocumentEmbeddings
```

```
In [ ]: from datasets import load_dataset, Split
        from transformers import (
            DataCollatorForLanguageModeling,
            BertModel, BertTokenizer, BertForSequenceClassification,
            DistilBertModel, DistilBertTokenizer,
            DistilBertForSequenceClassification,
            AutoModelForSequenceClassification,
            AutoModel, AutoTokenizer, AutoModelForCausalLM,
            Trainer, TrainingArguments, pipeline, TextClassificationPipeline
        )
        from transformers.pipelines import SUPPORTED_TASKS
```

Down the Yellow Brick Rabbit Hole

Left: "Alice and the Baby Pig" illustration by John Tenniel's, from "Alice's Adventure's in Wonderland" (1865).

Right: "Dorothy meets the Cowardly Lion" illustration by W.W. Denslow, from "The Wonderful Wizard of Oz" (1900)

Classification task

- Given a sentence we want to classify if this is written into *ALICE'S ADVENTURES IN WONDERLAN* or *The wonderful Wizard of Oz*.
- This is a Natural Language Processing (NLP) task.
- We explore how to handle word that are not numbers. How we can use them to learn and train a model and use it in various tasks.
- Tokenization, Embeddings, Bert, ChatGPT...
- Some references to understand what we do in Software Analysis with ML (you can find them also on our repository)

https://arxiv.org/pdf/2002.08155 (https://arxiv.org/pdf/2002.08155)

https://arxiv.org/abs/1803.09473 (https://arxiv.org/abs/1803.09473)

https://dl.acm.org/doi/pdf/10.1145/3460348 (https://dl.acm.org/doi/pdf/10.1145/3460348)

Building a Dataset

In [ ]:
```
localfolder = 'texts'
download_text(ALICE_URL, localfolder)
download_text(WIZARD_URL, localfolder)
```

In [ ]:
```
with open(os.path.join(localfolder, 'alice28-1476.txt'), 'r') as f:
    alice = ''.join(f.readlines()[104:3704])

with open(os.path.join(localfolder, 'wizoz10-1740.txt'), 'r') as f:
    wizard = ''.join(f.readlines()[310:5100])
```

In [ ]:
```python
print(alice[:500])
print('\n')
print(wizard[:500])
```

```
                ALICE'S ADVENTURES IN WONDERLAND

                          Lewis Carroll

               THE MILLENNIUM FULCRUM EDITION 2.8




                            CHAPTER I

                      Down the Rabbit-Hole


  Alice was beginning to get very tired of sitting by her sister
on the bank, and of having nothing to do:  once or twice she had
peeped into the book her sister was reading, but it had no
pictures or conversations in it, `and what is the use of a book,'
thought Alice `w


                    THE WONDERFUL WIZARD OF OZ


                          1.  The Cyclone


    Dorothy lived in the midst of the great Kansas prairies, with
Uncle Henry, who was a farmer, and Aunt Em, who was the farmer's
wife.  Their house was small, for the lumber to build it had to be
carried by wagon many miles.  There were four walls, a floor and a
roof, which made one room; and this room contained a rusty looking
cookstove, a cupboard for the dishes, a table, three or four
chairs, and th
```

You can explore the content of the downloaded text on Colab left panel [folder icon] and you could see that the initial part of both books contains useless text for our purposes.

We write a cfg file that contains the lines to be considered then we will proceed on identifying sentences in the books that will be our training dataset

In [ ]:
```python
text_cfg = """fname,start,end
alice28-1476.txt,104,3704
wizoz10-1740.txt,310,5100"""
bytes_written = open(os.path.join(localfolder, 'lines.cfg'), 'w').write(text_cfg)
```

Setting the goals

Our goal is to **transform** the books in CSV files so that they can be read as datasets.

Each line of the CSV will have a line for each sentence of the book, so that we could try to classify them, accordingly.

Something like:

```
sentence,source
"dsf sdf sdfsd sdfs, sdfs: dsfsdf, sdfs", Alice
"ipopsd iops siopiopi sdoiop sdifop", Alice
"mnmnb nmmnnj jj kjkj; \"jkj jk jk, ", Alice
"qweq qweq qweqw, qweqe ,\"qweqwe w,qwe,w\" ", Alice
```

Sentence Tokenization

You might have already met the words **token** and **tokenization**

- A **token** is a piece of text
- Tokenize a text means to split it into pieces (token), getting as result a list of **tokens**

The most common kind of pieces is a word, typically tokenizing a text usually means to split it into words, for instance, using the **white space** a separator.

```
In [ ]: sentence = "I'm following the white rabbit"
        tokens = sentence.split(' ')
        tokens
```

```
Out[ ]: ["I'm", 'following', 'the', 'white', 'rabbit']
```

What about *I'm*, it should be two tokens or just one?

IT DEPENDS :-)!

We will see later how to more properly perform tokenization at the word level.

For now, we are interested in sentence tokenization, our task is to classify sentences.

In order to do that, we use a library called NLTK (Natural Language Toolkit) and it `sent_tokenize()` method.

`punkt` is a pre-trained model for tokenization.

- It's based on the Punkt Sentence Tokenizer, a popular unsupervised method for sentence boundary detection.
- NLTK's punkt tokenizer can split text into sentences and further tokenize sentences into words.
- Language Support: punkt includes models trained for multiple languages (e.g., English, German, French).

```
In [ ]: import nltk
        from nltk.tokenize import sent_tokenize

        nltk.download('punkt')
        corpus_alice = sent_tokenize(alice)
        corpus_wizard = sent_tokenize(wizard)
        len(corpus_alice), len(corpus_wizard)
```

```
        [nltk_data] Downloading package punkt to /root/nltk_data...
        [nltk_data]   Package punkt is already up-to-date!
Out[ ]: (1612, 2240)
```

A **corpus** is a structured set of **Documents**.

A document can be a **sentence**, a **paragraph**, a **tweet** or a **book**.

In our case, the document is a **sentence**, so each book is actually a set of **sentences**, thus each book may be considered a corpus...

Its plural???

CORPORA

```
In [ ]: corpus_alice[2]
```

```
Out[ ]: 'There was nothing so VERY remarkable in that; nor did Alice\nthink it so VERY much out of the way to hear the
        Rabbit say to\nitself, `Oh dear!'
```

This includes `\n`, the sentence tokenizer only handles the sentence splitting and it doesn't clean the text.

```
In [ ]: corpus_wizard[30]
```

```
Out[ ]: '"There\'s a cyclone coming, Em," he called to his wife.'
```

The same here, we have " the quotation mark

Remember, we need to create CSV files with one sentence per line. Therefore, we need to:

- Clean the line breaks
- Define a quote char to wrap such that the original commas or semicolons are not misinteoreted as separation chars of the CSV
- add a second column (label) to identify the source

Then we will concatenate everything, shuffle the sentences before training a model

We expect to have something like this:

```
\"There's a cyclone coming, Em," he called to his wife.\,wizoz10-1740.txt
```

Where the **escape** character "\" is the quote char because is not present in any (novel) books.

Different choices has to be done when we will deal with code data.

```
In [ ]: def sentence_tokenize(source, quote_char='\\', sep_char=',',
                              include_header=True, include_source=True,
                              extensions=('txt'), **kwargs):
            nltk.download('punkt')
            # If source is a folder, goes through all files inside it
            # that match the desired extensions ('txt' by default)
            if os.path.isdir(source):
                filenames = [f for f in os.listdir(source)
                             if os.path.isfile(os.path.join(source, f)) and
                                os.path.splitext(f)[1][1:] in extensions]
            elif isinstance(source, str):
                filenames = [source]

            # If there is a configuration file, builds a dictionary with
            # the corresponding start and end lines of each text file
            config_file = os.path.join(source, 'lines.cfg')
            config = {}
            if os.path.exists(config_file):
                with open(config_file, 'r') as f:
                    rows = f.readlines()

                for r in rows[1:]:
                    fname, start, end = r.strip().split(',')
                    config.update({fname: (int(start), int(end))})

            new_fnames = []
            # For each file of text
            for fname in filenames:
                # If there's a start and end line for that file, use it
                try:
                    start, end = config[fname]
                except KeyError:
                    start = None
                    end = None

                # Opens the file, slices the configures lines (if any)
                # cleans line breaks and uses the sentence tokenizer
                with open(os.path.join(source, fname), 'r') as f:
                    contents = (''.join(f.readlines()[slice(start, end, None)])
                                .replace('\n', ' ').replace('\r', ''))
                corpus = sent_tokenize(contents, **kwargs)

                # Builds a CSV file containing tokenized sentences
                base = os.path.splitext(fname)[0]
                new_fname = f'{base}.sent.csv'
                new_fname = os.path.join(source, new_fname)
                with open(new_fname, 'w') as f:
                    # Header of the file
                    if include_header:
                        if include_source:
                            f.write('sentence,source\n')
                        else:
                            f.write('sentence\n')
                    # Writes one line for each sentence
                    for sentence in corpus:
                        if include_source:
                            f.write(f'{quote_char}{sentence}{quote_char}{sep_char}{fname}\n')
                        else:
                            f.write(f'{quote_char}{sentence}{quote_char}\n')
                new_fnames.append(new_fname)

            # Returns list of the newly generated CSV files
            return sorted(new_fnames)
```

The function above:

- takes a folder and goes through the file with the right extension
- it removes the lines based on a `.cfg` file (if any)
- it applies the sentence tokenizer to each file
- it generates the corresponding CSV files of sentences using the configured `quote_char` and `sep_char`
- it names the CSV files for each original file by dropping the extension and appending `.sent.csv` to it.

```
In [ ]: new_fnames = sentence_tokenize(localfolder)
        new_fnames

        [nltk_data] Downloading package punkt to /root/nltk_data...
        [nltk_data]   Package punkt is already up-to-date!

Out[ ]: ['texts/alice28-1476.sent.csv', 'texts/wizoz10-1740.sent.csv']
```

Each CSV file contains the sentences of a book and we will use them to build our dataset (concatenating and shuffling)

HuggingFace's Dataset

Loading a Dataset

Instead of using regular PyTorch `Dataset` we use **Hugging face*** `Dataset`.

In order to accomplish our task, we will use a pre-trained method coming from hugging face.

Hugging face

Hugging Face is a company and open-source platform focused on making advanced machine learning, particularly natural language processing (NLP), more accessible and easier to use. It’s widely recognized for its transformer library and model hub, which provide pre-trained models and tools to streamline the use of state-of-the-art AI models in applications.

Here are the key aspects of Hugging Face:

1. Transformers Library Hugging Face’s transformers library is one of the most popular libraries for NLP and beyond, offering access to hundreds of pre-trained models like BERT, GPT, RoBERTa, T5, and many others. It allows users to load and fine-tune these models for tasks like text classification, translation, summarization, question answering, text generation, and more. The library supports multiple deep learning frameworks, primarily PyTorch and TensorFlow, and includes APIs that make it easy to switch between them.
2. Model Hub The Hugging Face Model Hub is a repository and sharing platform for machine learning models. It hosts thousands of pre-trained models contributed by both Hugging Face and the community. Users can find, share, and download models for various tasks (not limited to NLP, but also including computer vision, audio, and more). The hub allows seamless model access via model IDs that can be loaded directly into the transformers library, enabling rapid experimentation and deployment.
3. Datasets Library Hugging Face’s datasets library provides a large collection of datasets for NLP, computer vision, and audio tasks. It includes popular datasets like IMDB, SQuAD, and ImageNet. It supports streaming, efficient data handling, and pre-processing tools, making it easier to train models on large datasets.

```
In [ ]: from datasets import load_dataset, Split

        dataset = load_dataset(path='csv', data_files=new_fnames, quotechar='\\', split=Split.TRAIN)
```

```
In [ ]: type(dataset)

Out[ ]:  datasets.arrow_dataset.Dataset
         def __init__(arrow_table: Table, info: Optional[DatasetInfo]=None, split: Optional[NamedSplit]=None,
         indices_table: Optional[Table]=None, fingerprint: Optional[str]=None)

        A Dataset backed by an Arrow table.
```

What we have done?

We created a hugging face's dataset, specifying:

1. the type of script we are using to process the data `path='csv'` (Yes, a bit misleading)
2. the path where the files are located `data_files=new_fnames`
3. the quotation mark
4. the split we are generating (TRAIN)

Attributes

```
In [ ]: dataset.features, dataset.num_columns, dataset.shape
```

```
Out[ ]: ({'sentence': Value(dtype='string', id=None),
          'source': Value(dtype='string', id=None)},
         2,
         (3852, 2))
```

Our dataset contains **2** columns (sentence, source) and there are 3852 sentences on it.

```
In [ ]: dataset[2]
```

```
Out[ ]: {'sentence': 'There was nothing so VERY remarkable in that; nor did Alice think it so VERY much out of the way
        to hear the Rabbit say to itself, `Oh dear!',
         'source': 'alice28-1476.txt'}
```

```
In [ ]: dataset['source'][:3] ## we didnt' shuffle yet
```

```
Out[ ]: ['alice28-1476.txt', 'alice28-1476.txt', 'alice28-1476.txt']
```

Methods of `Dataset`

- unique, to compute for instance, the unique sources
- map, to create new columns by using a function

```
In [ ]: dataset.unique('source')
```

```
Out[ ]: ['alice28-1476.txt', 'wizoz10-1740.txt']
```

```
In [ ]: def is_alice_label(row):
            is_alice = int(row['source'] == 'alice28-1476.txt')
            return {'labels': is_alice}

        dataset = dataset.map(is_alice_label)
```

We create a new column for label and it is already applied to the dataset

```
In [ ]: dataset[2]
```

```
Out[ ]: {'sentence': 'There was nothing so VERY remarkable in that; nor did Alice think it so VERY much out of the way
        to hear the Rabbit say to itself, `Oh dear!',
         'source': 'alice28-1476.txt',
         'labels': 1}
```

```
In [ ]: shuffled_dataset = dataset.shuffle(seed=42)
In [ ]: split_dataset = shuffled_dataset.train_test_split(test_size=0.2)
        split_dataset
```

```
Out[ ]: DatasetDict({
            train: Dataset({
                features: ['sentence', 'source', 'labels'],
                num_rows: 3081
            })
            test: Dataset({
                features: ['sentence', 'source', 'labels'],
                num_rows: 771
            })
        })
```

```
In [ ]: train_dataset, test_dataset = split_dataset['train'], split_dataset['test']
```

```
In [ ]: train_dataset.features
```

```
Out[ ]: {'sentence': Value(dtype='string', id=None),
         'source': Value(dtype='string', id=None),
         'labels': Value(dtype='int64', id=None)}
```

```
In [ ]:
```

Word Tokenization

Now we have the dataset structured with all the information we need.

But the basic bricks of NLP tasks are the words, and we need to process them...

We already see a simple word tokenizer that uses the white space as separator, but this doesn't work well as we want.

```
In [ ]: sentence = "I'm following the white rabbit"
        tokens = sentence.split(' ')
        tokens
```

```
Out[ ]: ["I'm", 'following', 'the', 'white', 'rabbit']
```

We can use another library `gensim` that can help us on this task.

We could use the NLKT tokenizer, but Gensim includes interesting tools that we can use later, but doesn't have a sentence tokenizer... and also it is a way to know different libraries :-)

```
In [ ]: from gensim.parsing.preprocessing import *

        preprocess_string(sentence)
```

```
Out[ ]: ['follow', 'white', 'rabbit']
```

`def preprocess_string(s, filters=DEFAULT_FILTERS)` Apply list of chosen filters to s.

Default list of filters:

```
~gensim.parsing.preprocessing.strip_tags,
~gensim.parsing.preprocessing.strip_punctuation,
~gensim.parsing.preprocessing.strip_multiple_whitespaces,
~gensim.parsing.preprocessing.strip_numeric,
~gensim.parsing.preprocessing.remove_stopwords,
~gensim.parsing.preprocessing.strip_short,
~gensim.parsing.preprocessing.stem_text.
```

Parameters s : str filters: list of functions, optional

Returns list of str Processed strings (cleaned).

We keep the list of filters limited: lower case, tags, punctuactions, multiple white spaces and numeric.

You can play on your own with the filters and check the obtained results...

```
In [ ]: filters = [lambda x: x.lower(), strip_tags, strip_punctuation, strip_multiple_whitespaces, strip_numeric]
        preprocess_string(sentence, filters=filters)
```

```
Out[ ]: ['i', 'm', 'following', 'the', 'white', 'rabbit']
```

Another option is to use `simple_preprocess` method that limit the filters to:

- lower case
- remove tokens too short (less than 3 chars)
- remove tokens too long (more than 15 chars)

```
In [ ]: from gensim.utils import simple_preprocess

        tokens = simple_preprocess(sentence)
        tokens
```

```
Out[ ]: ['following', 'the', 'white', 'rabbit']
```

Vocabulary

Once we have the token, a list of words, we can build our vocabulary, a list of unique words that appear in the text corpora.

```
In [ ]: sentences = train_dataset['sentence'] ## we take the column sentence
        tokens = [simple_preprocess(sent) for sent in sentences] # for each sentence we extract the tokens
        tokens[0]
```

```
Out[ ]: ['then',
         'he',
         'said',
         'do',
         'you',
         'suppose',
         'oz',
         'could',
         'give',
         'me',
         'heart']
```

```
In [ ]: '''
        Once we have the tokens we can build our vocabulary, a list of unique words that appear in the text corpora.
        '''

        from gensim import corpora

        dictionary = corpora.Dictionary(tokens)
        print(dictionary)

        Dictionary<3674 unique tokens: ['could', 'do', 'give', 'he', 'heart']...>
```

The Dictionary object computes some specific attributes

```
In [ ]: dictionary.num_docs
```

```
Out[ ]: 3081
```

```
In [ ]: dictionary.num_pos # processed words
```

```
Out[ ]: 50766
```

```
In [ ]: dictionary.token2id
```

```
Out[ ]: {'could': 0,
         'do': 1,
         'give': 2,
         'he': 3,
         'heart': 4,
         'me': 5,
         'oz': 6,
         'said': 7,
         'suppose': 8,
         'then': 9,
         'you': 10,
         'at': 11,
         'don': 12,
         'gryphon': 13,
         'if': 14,
         'is': 15,
         'know': 16,
         'look': 17,
         'picture': 18,
         'the': 19,
         'what': 20,
         'against': 21,
         'alice': 22,
         'and': 23,
         'as': 24,
         'been': 25,
         'buttercup': 26,
         'fanned': 27,
         'have': 28,
         'herself': 29,
         'it': 30,
         'leant': 31,
         'leaves': 32,
         'liked': 33,
         'much': 34,
         'of': 35,
         'one': 36,
         'only': 37,
         'rest': 38,
         'right': 39,
         'she': 40,
         'should': 41,
         'size': 42,
         'teaching': 43,
         'to': 44,
         'tricks': 45,
         'very': 46,
         'with': 47,
         'added': 48,
         'any': 49,
         'but': 50,
         'grow': 51,
         'grown': 52,
         'here': 53,
         'in': 54,
         'least': 55,
         'll': 56,
         'more': 57,
         'no': 58,
         'now': 59,
         'room': 60,
         'sorrowful': 61,
         'there': 62,
         'tone': 63,
         'up': 64,
         'when': 65,
         'write': 66,
         'course': 67,
         'mock': 68,
         'turtle': 69,
         'was': 70,
         'that': 71,
         'again': 72,
         'dorothy': 73,
         'last': 74,
         'looked': 75,
         'they': 76,
         'didn': 77,
         'for': 78,
         'great': 79,
         'laid': 80,
         'many': 81,
         'table': 82,
         'than': 83,
         'three': 84,
         'your': 85,
         'before': 86,
         'cowardly': 87,
         'did': 88,
         'drink': 89,
         'like': 90,
         'lion': 91,
         'not': 92,
         'placing': 93,
         'sniffed': 94,
         'this': 95,
         'who': 96,
         'wizard': 97,
         'be': 98,
         'cannot': 99,
         'cross': 100,
         'desert': 101,
         'glad': 102,
         'our': 103,
         'power': 104,
         'serve': 105,
         'shall': 106,
         'way': 107,
         'we': 108,
         'answer': 109,
         'confused': 110,
         'dormouse': 111,
         'go': 112,
         'interrupting': 113,
         'let': 114,
         'on': 115,
         'poor': 116,
         'so': 117,
         'some': 118,
         'time': 119,
         'without': 120,
         'bear': 121,
         'catch': 122,
         'coming': 123,
         'down': 124,
         'hold': 125,
         'loose': 126,
         'mind': 127,
         'oh': 128,
         'roof': 129,
         'rope': 130,
         'slate': 131,
         'will': 132,
         'ah': 133,
         'away': 134,
         'melt': 135,
         'minute': 136,
         'cat': 137,
         'cats': 138,
         'dinah': 139,
         'fancy': 140,
         'her': 141,
         'see': 142,
         'show': 143,
         'take': 144,
         'think': 145,
         'wish': 146,
         'yet': 147,
         'command': 148,
         'creatures': 149,
         'person': 150,
         'strange': 151,
         'these': 152,
         'times': 153,
         'better': 154,
         'can': 155,
         'follow': 156,
         'had': 157,
         'politely': 158,
         'quite': 159,
         'say': 160,
         'understand': 161,
         'written': 162,
         'knows': 163,
                    'little': 164,
                    'such': 165,
                    'hastily': 166,
                    'mean': 167,
                    'replied': 168,
                    'same': 169,
                    'thing': 170,
                    'an': 171,
                    'anything': 172,
                    'aunt': 173,
                    'by': 174,
                    'came': 175,
                    'child': 176,
                    'ears': 177,
                    'em': 178,
                    'find': 179,
                    'first': 180,
                    'girl': 181,
                    'hand': 182,
                    'laugh': 183,
                    'laughter': 184,
                    'merry': 185,
                    'orphan': 186,
                    'press': 187,
                    'reached': 188,
                    'scream': 189,
                    'startled': 190,
                    'still': 191,
                    'upon': 192,
                    'voice': 193,
                    'whenever': 194,
                    'wonder': 195,
                    'would': 196,
                    'cry': 197,
                    'fright': 198,
                    'gave': 199,
                    'leg': 200,
                    'made': 201,
                    'out': 202,
                    'tin': 203,
                    'tinsmith': 204,
                    'went': 205,
                    'about': 206,
                    'after': 207,
                    'air': 208,
                    'appearance': 209,
                    'being': 210,
                    'cheshire': 211,
                    'curious': 212,
                    'escape': 213,
                    'get': 214,
                    'grin': 215,
                    'looking': 216,
                    'noticed': 217,
                    'or': 218,
                    'puzzled': 219,
                    'seen': 220,
                    'somebody': 221,
                    'talk': 222,
                    'two': 223,
                    'watching': 224,
                    'whether': 225,
                    'wondering': 226,
                    'across': 227,
                    'axe': 228,
                    'began': 229,
                    'both': 230,
                    'bottom': 231,
                    'brutes': 232,
                    'carrying': 233,
                    'crash': 234,
                    'dashed': 235,
                    'fell': 236,
                    'gulf': 237,
                    'his': 238,
                    'into': 239,
                    'just': 240,
                    'kalidahs': 241,
                    'nearly': 242,
                    'once': 243,
                    'pieces': 244,
                    'rocks': 245,
                    'sharp': 246,
                    'snarling': 247,
                    'tree': 248,
                    'ugly': 249,
                    'use': 250,
                    'were': 251,
                    'woodman': 252,
                    'clapping': 253,
                    'clever': 254,
                    'day': 255,
                    'general': 256,
                    'hands': 257,
                    'king': 258,
                    'really': 259,
                    'apple': 260,
                    'bottle': 261,
                    'buttered': 262,
                    'cherry': 263,
                    'custard': 264,
                    'fact': 265,
                    'finding': 266,
                    'finished': 267,
                    'flavour': 268,
                    'hot': 269,
                    'however': 270,
                    'marked': 271,
                    'mixed': 272,
                    'nice': 273,
                    'off': 274,
                    'pine': 275,
                    'poison': 276,
                    'roast': 277,
                    'soon': 278,
                    'sort': 279,
                    'tart': 280,
                    'taste': 281,
                    'toast': 282,
                    'toffee': 283,
                    'turkey': 284,
                    'ventured': 285,
                    'boldly': 286,
                    'forward': 287,
                    'hill': 288,
                    'must': 289,
                    'over': 290,
                    'pass': 291,
                    'sorry': 292,
                    'walked': 293,
                    'love': 294,
                    'therefore': 295,
                    'begun': 296,
                    'friend': 297,
                    'old': 298,
                    'pleaded': 299,
                    'woman': 300,
                    'home': 301,
                    'magic': 302,
                    'my': 303,
                    'send': 304,
                    'something': 305,
                    'all': 306,
                    'baby': 307,
                    'believed': 308,
                    'bells': 309,
                    'boy': 310,
                    'busy': 311,
                    'cattle': 312,
                    'change': 313,
                    'clamour': 314,
                    'closed': 315,
                    'cries': 316,
                    'distance': 317,
                    'dull': 318,
                    'eyes': 319,
                    'farm': 320,
                    'grass': 321,
                    'half': 322,
                    'heavy': 323,
                    'knew': 324,
                    'lowing': 325,
                    'noises': 326,
                    'open': 327,
                    'other': 328,
                    'place': 329,
                    'pool': 330,
                    'queen': 331,
                    'queer': 332,
                    'rattling': 333,
                    'reality': 334,
                    'reeds': 335,
                    'rippling': 336,
                    'rustling': 337,
                    'sat': 338,
                    'sheep': 339,
                    'shepherd': 340,
                    'shriek': 341,
                    'shrill': 342,
                    'sneeze': 343,
                    'sobs': 344,
                    'teacups': 345,
                    'them': 346,
                    'though': 347,
                    'thy': 348,
                    'tinkling': 349,
                    'waving': 350,
                    'while': 351,
                    'wind': 352,
                    'wonderland': 353,
                    'yard': 354,
                    'head': 355,
                    'impatiently': 356,
                    'its': 357,
                    'mouse': 358,
                    'quicker': 359,
                    'shook': 360,
                    'difficulties': 361,
                    'settling': 362,
                    'small': 363,
                    'mended': 364,
                    'broken': 365,
                    'ceiling': 366,
                    'drunk': 367,
                    'expected': 368,
                    'found': 369,
                    'from': 370,
                    'indeed': 371,
                    'neck': 372,
                    'pressing': 373,
                    'save': 374,
                    'sooner': 375,
                    'stoop': 376,
                    'arms': 377,
                    'bye': 378,
                    'dog': 379,
                    'except': 380,
                    'flights': 381,
                    'followed': 382,
                    'friends': 383,
                    'front': 384,
                    'good': 385,
                    'green': 386,
                    'palace': 387,
                    'passages': 388,
                    'seven': 389,
                    'stairs': 390,
                    'taking': 391,
                    'through': 392,
                    'toto': 393,
                    'until': 394,
                    'allow': 395,
                    'anyone': 396,
                    'belongs': 397,
                    'itself': 398,
                    'rock': 399,
                    'showed': 400,
                    'us': 401,
                    'kind': 402,
                    'scarecrow': 403,
                    'ask': 404,
                    'might': 405,
                    'question': 406,
                    'venture': 407,
                    'agree': 408,
                    'duchess': 409,
                    'everything': 410,
                    'large': 411,
                    'mine': 412,
                    'mustard': 413,
                    'near': 414,
                    'ready': 415,
                    'seemed': 416,
                    'dear': 417,
                    'paws': 418,
                    'afraid': 419,
                    'am': 420,
                    'world': 421,
                    'also': 422,
                    'brains': 423,
                    'having': 424,
                    'rather': 425,
                    'tried': 426,
                    'bright': 427,
                    'cakes': 428,
                    'floor': 429,
                    'idea': 430,
                    'lay': 431,
                    'pebbles': 432,
                    'surprise': 433,
                    'turning': 434,
                    'center': 435,
                    'cyclone': 436,
                    'exact': 437,
                    'house': 438,
                    'met': 439,
                    'north': 440,
                    'south': 441,
                    'stood': 442,
                    'where': 443,
                    'winds': 444,
                    'along': 445,
                    'come': 446,
                    'company': 447,
                    'heartily': 448,
                    'pleased': 449,
                    'easy': 450,
                    'left': 451,
                    'race': 452,
                    'running': 453,
                    'leave': 454,
                    'became': 455,
                    'birds': 456,
                    'brightly': 457,
                    'carpeted': 458,
                    'colored': 459,
                    'flowers': 460,
                    'ground': 461,
                    'listening': 462,
                    'lovely': 463,
                    'singing': 464,
                    'thick': 465,
                    'which': 466,
                    'why': 467,
                    'foot': 468,
                    'standing': 469,
                    'bed': 470,
                    'door': 471,
                    'heels': 472,
                    'opened': 473,
                    'ran': 474,
                    'sprang': 475,
                    'bill': 476,
                    'end': 477,
                    'extra': 478,
                    'french': 479,
                    'music': 480,
                    'ours': 481,
                    'washing': 482,
                    'always': 483,
                    'blame': 484,
                    'others': 485,
                    'crimson': 486,
                    'crown': 487,
                    'cushion': 488,
                    'grand': 489,
                    'hearts': 490,
                    'knave': 491,
                    'procession': 492,
                    'velvet': 493,
                    'footman': 494,
                    'knocking': 495,
                    'reasons': 496,
                    'begin': 497,
                    'majesty': 498,
                    'please': 499,
                    'fur': 500,
                    'whiskers': 501,
                    'asked': 502,
                    'around': 503,
                    'barking': 504,
                    'butterflies': 505,
                    'chased': 506,
                    'country': 507,
                    'delight': 508,
                    'fresh': 509,
                    'joy': 510,
                    'merrily': 511,
                    'moths': 512,
                    'pure': 513,
                    'side': 514,
                    'tail': 515,
                    'whisked': 516,
                    'sing': 517,
                    'kiss': 518,
                    'likes': 519,
                    'may': 520,
                    'stock': 521,
                    'barrowful': 522,
                    'saw': 523,
                    'sharply': 524,
                    'telling': 525,
                    'truth': 526,
                    'back': 527,
                    'behold': 528,
                    'carry': 529,
                    'castle': 530,
                    'clean': 531,
                    'picked': 532,
                    'straw': 533,
                    'stuffed': 534,
                    'winkies': 535,
                    'hatter': 536,
                    'ever': 537,
                    'mournful': 538,
                    'since': 539,
                    'won': 540,
                    'afford': 541,
                    'couldn': 542,
                    'learn': 543,
                    'song': 544,
                    'behind': 545,
                    'broke': 546,
                    'grazed': 547,
                    'him': 548,
                    'moment': 549,
                    'nose': 550,
                    'plate': 551,
                    'remarked': 552,
                    'sit': 553,
                    'skimming': 554,
                    'straight': 555,
                    'till': 556,
                    'tomorrow': 557,
                    'trees': 558,
                    'tears': 559,
                    'glass': 560,
                    'sounds': 561,
                    'big': 562,
                    'building': 563,
                    'city': 564,
                    'exactly': 565,
                    'gates': 566,
                    'guardian': 567,
                    'led': 568,
                    'middle': 569,
                    'streets': 570,
                    'frightened': 571,
                    'pile': 572,
                    'shaggy': 573,
                    'told': 574,
                    'wolves': 575,
                    'brick': 576,
                    'current': 577,
                    'downstream': 578,
                    'farther': 579,
                    'got': 580,
                    'raft': 581,
                    'river': 582,
                    'road': 583,
                    'swept': 584,
                    'swift': 585,
                    'well': 586,
                    'yellow': 587,
                    'exclaimed': 588,
                    'happy': 589,
                    'never': 590,
                    'patch': 591,
                    'inquired': 592,
                    'dinner': 593,
                    'fellow': 594,
                    'forest': 595,
                    'gayelette': 596,
                    'grandfather': 597,
                    'joke': 598,
                    'lived': 599,
                    'loved': 600,
                    'monkeys': 601,
                    'winged': 602,
                    'among': 603,
                    'beating': 604,
                    'curving': 605,
                    'dive': 606,
                    'draw': 607,
                    'face': 608,
                    'flown': 609,
                    'going': 610,
                    'graceful': 611,
                    'hiss': 612,
                    'hurry': 613,
                    'nothing': 614,
                    'pigeon': 615,
                    'succeeded': 616,
                    'tops': 617,
                    'under': 618,
                    'violently': 619,
                    'wandering': 620,
                    'wings': 621,
                    'zigzag': 622,
                    'killing': 623,
                    'smell': 624,
                    'fall': 625,
                    'gown': 626,
                    'silk': 627,
                    'spot': 628,
                    'sighing': 629,
                    'demanded': 630,
                    'badly': 631,
                    'long': 632,
                    'poles': 633,
                    'push': 634,
                    'steady': 635,
                    'stepped': 636,
                    'their': 637,
                    'tipped': 638,
                    'water': 639,
                    'emerald': 640,
                    'luck': 641,
                    'pulled': 642,
                    'stake': 643,
                    'sure': 644,
                    'hung': 645,
                    'wire': 646,
                    'beheaded': 647,
                    'shan': 648,
                    'are': 649,
                    'crows': 650,
                    'dare': 651,
                    'nearer': 652,
                    'scarecrows': 653,
                    'wicked': 654,
                    'witch': 655,
                    'fitted': 656,
                    'own': 657,
                    'hear': 658,
                    'dance': 659,
                    'feeling': 660,
                    'interesting': 661,
                    'thank': 662,
                    'watch': 663,
                    'whiting': 664,
                    'brought': 665,
                    'fasten': 666,
                    'mice': 667,
                    'strings': 668,
                    'truck': 669,
                    'using': 670,
                    'eagerly': 671,
                    'tell': 672,
                    'climbed': 673,
                    'holding': 674,
                    'mane': 675,
                    'next': 676,
                    'thought': 677,
                    'tightly': 678,
                    'took': 679,
                    'call': 680,
                    'certainly': 681,
                    'sad': 682,
                    'destroyed': 683,
                    'tied': 684,
                    'close': 685,
                    'courage': 686,
                    'each': 687,
                    'gained': 688,
                    'mouths': 689,
                    'nervous': 690,
                    'wide': 691,
                    'civilized': 692,
                    'cut': 693,
                    'has': 694,
                    'land': 695,
                    'certain': 696,
                    'fly': 697,
                    'baskets': 698,
                    'bend': 699,
                    'bringing': 700,
                    'dents': 701,
                    'shape': 702,
                    'solder': 703,
                    'straighten': 704,
                    'those': 705,
                    'tinsmiths': 706,
                    'together': 707,
                    'tools': 708,
                    'dizzy': 709,
                    'hurt': 710,
                    'surprised': 711,
                    'wow': 712,
                    'answered': 713,
                    'plenty': 714,
                    'croquet': 715,
                    'invitation': 716,
                    'play': 717,
                    'yes': 718,
                    'bats': 719,
                    'eat': 720,
                    'sometimes': 721,
                    'contented': 722,
                    'continued': 723,
                    'live': 724,
                    'declared': 725,
                    'easily': 726,
                    'enough': 727,
                    'man': 728,
                    'beautiful': 729,
                    'bracelet': 730,
                    'collar': 731,
                    'determined': 732,
                    'diamonds': 733,
                    'gold': 734,
                    'golden': 735,
                    'headed': 736,
                    'inlaid': 737,
                    'jewels': 738,
                    'keep': 739,
                    'offered': 740,
                    'oil': 741,
                    'precious': 742,
                    'presented': 743,
                    'set': 744,
                    'silver': 745,
                    'stick': 746,
                    'studded': 747,
                    'stumbling': 748,
                    'walking': 749,
                    'decided': 750,
                    'late': 751,
                    'waited': 752,
                    'quadlings': 753,
                    'full': 754,
                    'deep': 755,
                    'grew': 756,
                    'touch': 757,
                    'almost': 758,
                    'different': 759,
                    'remember': 760,
                    'idiot': 761,
                    'bit': 762,
                    'bite': 763,
                    'flamingoes': 764,
                    'true': 765,
                    'presently': 766,
                    'ventriloquist': 767,
                    'caterpillar': 768,
                    'encouraging': 769,
                    'hare': 770,
                    'march': 771,
                    'wine': 772,
                    'because': 773,
                    'blue': 774,
                    'bodices': 775,
                    'breeches': 776,
                    'buckles': 777,
                    'caps': 778,
                    'cheeks': 779,
                    'clowns': 780,
                    'crowns': 781,
                    'doublets': 782,
                    'dressed': 783,
                    'ermine': 784,
                    'frocks': 785,
                    'funny': 786,
                    'gorgeous': 787,
                    'gowns': 788,
                    'heads': 789,
                    'jeweled': 790,
                    'knee': 791,
                    'milkmaids': 792,
                    'most': 793,
                    'pink': 794,
                    'pointed': 795,
                    'princes': 796,
                    'princesses': 797,
                    'purple': 798,
                    'red': 799,
                    'robes': 800,
                    'round': 801,
                    'ruffled': 802,
                    'satin': 803,
                    'shepherdesses': 804,
                    'shepherds': 805,
                    'shoes': 806,
                    'spots': 807,
                    'stripes': 808,
                    'tall': 809,
                    'wearing': 810,
                    'basket': 811,
                    'bread': 812,
                    'cloth': 813,
                    'cupboard': 814,
                    'filled': 815,
                    'laying': 816,
                    'top': 817,
                    'white': 818,
                    'awfully': 819,
                    'fast': 820,
                    'roared': 821,
                    'run': 822,
                    'scared': 823,
                    've': 824,
                    'east': 825,
                    'fortunately': 826,
                    'harm': 827,
                    'powerful': 828,
                    'surely': 829,
                    'terribly': 830,
                    'themselves': 831,
                    'west': 832,
                    'witches': 833,
                    'bondage': 834,
                    'called': 835,
                    'delighted': 836,
                    'free': 837,
                    'help': 838,
                    'rescue': 839,
                    'beat': 840,
                    'sadly': 841,
                    'advantage': 842,
                    'deal': 843,
                    'jug': 844,
                    'milk': 845,
                    'upset': 846,
                    'worse': 847,
                    'clothes': 848,
                    'cried': 849,
                    'fine': 850,
                    'spotted': 851,
                    'swim': 852,
                    'dangerous': 853,
                    'joints': 854,
                    'kissed': 855,
                    'weeping': 856,
                    'fortune': 857,
                    'quelala': 858,
                    'spoiled': 859,
                    'too': 860,
                    'wise': 861,
                    'glasses': 862,
                    'bees': 863,
                    'commanded': 864,
                    'flew': 865,
                    'rapidly': 866,
                    'turned': 867,
                    'form': 868,
                    'line': 869,
                    'lines': 870,
                    'sea': 871,
                    'shore': 872,
                    'anxiously': 873,
                    'kansas': 874,
                    'lives': 875,
                    'decidedly': 876,
                    'sky': 877,
                    'speaking': 878,
                    'uncivil': 879,
                    'bell': 880,
                    'fastened': 881,
                    'ribbon': 882,
                    'things': 883,
                    'forgetting': 884,
                    'rabbit': 885,
                    'reason': 886,
                    'thousand': 887,
                    'trembled': 888,
                    'lory': 889,
                    'fear': 890,
                    'curtsy': 891,
                    'friendly': 892,
                    'gravely': 893,
                    'nodded': 894,
                    'does': 895,
                    'maiden': 896,
                    'marry': 897,
                    'munchkin': 898,
                    'backs': 899,
                    'children': 900,
                    'courtiers': 901,
'faces': 902,
'gardeners': 903,
'lying': 904,
'pack': 905,
'pattern': 906,
'pointing': 907,
'rosetree': 908,
'soldiers': 909,
'beast': 910,
'kill': 911,
'heard': 912,
'uglification': 913,
'another': 914,
'common': 915,
'brave': 916,
'deed': 917,
'done': 918,
'life': 919,
'saving': 920,
'beautifully': 921,
'book': 922,
'hoping': 923,
'key': 924,
'label': 925,
'letters': 926,
'paper': 927,
'people': 928,
'printed': 929,
'rate': 930,
'rules': 931,
'shutting': 932,
'telescopes': 933,
'waiting': 934,
'words': 935,
'prizes': 936,
'beg': 937,
'speak': 938,
'beasts': 939,
'bowed': 940,
'promised': 941,
'rule': 942,
'safely': 943,
'farmhouse': 944,
'knocked': 945,
'sized': 946,
'obey': 947,
'willing': 948,
'cornfield': 949,
'lifted': 950,
'pole': 951,
'absurd': 952,
'grave': 953,
'simply': 954,
'solemn': 955,
'thimble': 956,
'whole': 957,
'how': 958,
'sudden': 959,
'journey': 960,
'cap': 961,
'hours': 962,
'nine': 963,
'ten': 964,
'crust': 965,
'dish': 966,
'disobey': 967,
'editions': 968,
'eye': 969,
'felt': 970,
'follows': 971,
'garden': 972,
'gravy': 973,
'later': 974,
'meat': 975,
'owl': 976,
'panther': 977,
'passed': 978,
'pie': 979,
'share': 980,
'sharing': 981,
'treat': 982,
'trembling': 983,
'wrong': 984,
'climb': 985,
'ditch': 986,
'neither': 987,
'zik': 988,
'ziz': 989,
'zuz': 990,
'zy': 991,
'bad': 992,
'capital': 993,
'london': 994,
'paris': 995,
'rome': 996,
'cow': 997,
'frightening': 998,
'empty': 999,
...}
```

```
In [ ]: vocab = list(dictionary.token2id.keys())
        vocab[:5]

Out[ ]: ['could', 'do', 'give', 'he', 'heart']
```

In [ ]:
```
#collection frequencies, how many times a given token appears in the corpora
dictionary.cfs
```

```
Out[ ]: {9: 177,
         3: 465,
         7: 625,
         1: 161,
         10: 727,
         8: 23,
         6: 129,
         0: 162,
         2: 43,
         5: 220,
         4: 53,
         14: 185,
         12: 75,
         16: 116,
         20: 192,
         13: 42,
         15: 290,
         17: 38,
         11: 369,
         19: 3654,
         18: 1,
         22: 331,
         24: 484,
         40: 755,
         31: 1,
         21: 12,
         26: 1,
         44: 1446,
         38: 20,
         29: 72,
         23: 2048,
         27: 1,
         47: 359,
         36: 176,
         35: 1081,
         32: 10,
         41: 68,
         28: 202,
         33: 7,
         43: 1,
         30: 837,
         45: 2,
         46: 175,
         34: 72,
         37: 83,
         25: 84,
         39: 45,
         42: 13,
         65: 194,
         51: 8,
         64: 161,
         56: 55,
         66: 5,
         50: 386,
         52: 10,
         59: 122,
         48: 24,
         54: 679,
         61: 2,
         63: 33,
         55: 8,
         62: 187,
         58: 166,
         60: 60,
         49: 71,
         57: 69,
         53: 69,
         67: 42,
         70: 684,
         68: 46,
         69: 48,
         71: 576,
         74: 52,
         76: 424,
         75: 80,
         73: 294,
         72: 129,
         77: 18,
         85: 101,
         82: 22,
         80: 3,
         78: 379,
         79: 141,
         81: 49,
         83: 61,
         84: 42,
         93: 1,
         95: 218,
         86: 87,
         87: 19,
         91: 132,
         96: 142,
         94: 2,
         88: 117,
         92: 317,
         90: 109,
         97: 29,
         89: 11,
         108: 147,
         106: 91,
         98: 263,
         102: 27,
         105: 3,
         107: 102,
         103: 31,
         104: 12,
         99: 29,
         100: 17,
         101: 13,
         109: 15,
         117: 360,
         110: 4,
         116: 34,
         114: 40,
         111: 32,
         112: 96,
         115: 279,
         118: 68,
         119: 87,
         120: 32,
         113: 2,
         122: 3,
         125: 13,
         130: 3,
         132: 141,
         129: 9,
         121: 5,
         127: 25,
         126: 1,
         131: 2,
         128: 69,
         123: 17,
         124: 157,
         133: 6,
         136: 20,
         135: 2,
         134: 74,
         147: 32,
         146: 43,
         143: 8,
         137: 28,
         139: 12,
         145: 75,
         144: 38,
         140: 6,
         138: 9,
         142: 132,
         141: 505,
         150: 8,
         148: 8,
         152: 45,
         151: 19,
         149: 12,
         153: 17,
         161: 11,
         154: 21,
         158: 9,
         157: 375,
         162: 8,
         155: 115,
         159: 80,
         156: 5,
         160: 52,
         163: 6,
         165: 42,
         164: 214,
         166: 14,
         168: 64,
         167: 10,
         169: 31,
         170: 51,
         171: 72,
         186: 1,
         180: 76,
         175: 120,
         173: 22,
         178: 23,
         190: 4,
         174: 117,
         176: 17,
         184: 4,
         196: 159,
         189: 3,
         187: 1,
         182: 25,
         192: 89,
         194: 6,
         185: 1,
         193: 65,
         188: 11,
         177: 10,
         191: 28,
         181: 75,
         195: 23,
         179: 45,
         172: 32,
         183: 6,
         199: 34,
         197: 13,
         198: 4,
         205: 105,
         204: 5,
         201: 98,
         200: 9,
         202: 183,
         203: 113,
         216: 41,
         206: 111,
         213: 5,
         226: 7,
         225: 19,
         214: 107,
         210: 32,
         220: 26,
         217: 10,
         212: 17,
         209: 1,
         208: 41,
         219: 10,
         207: 93,
         224: 7,
         218: 98,
         223: 49,
         215: 4,
         211: 6,
         221: 7,
         222: 13,
         252: 145,
         229: 69,
         250: 27,
         238: 254,
         228: 23,
         243: 69,
         240: 85,
         241: 4,
         251: 236,
         242: 23,
         227: 14,
         248: 29,
         236: 25,
         234: 7,
         239: 121,
         237: 2,
         233: 3,
         249: 4,
         247: 2,
         232: 1,
         230: 23,
         235: 5,
         244: 13,
         246: 15,
         245: 4,
         231: 11,
         256: 3,
         253: 2,
         257: 22,
         259: 29,
         254: 2,
         258: 77,
         255: 54,
         270: 23,
         261: 11,
         271: 6,
         276: 4,
         285: 4,
         281: 2,
         266: 4,
         273: 6,
         265: 8,
         279: 19,
         272: 2,
         268: 1,
         263: 1,
         280: 1,
         264: 1,
         275: 1,
         260: 1,
         277: 1,
         284: 1,
         283: 1,
         269: 13,
         262: 1,
         282: 1,
         278: 55,
         267: 19,
         274: 84,
         292: 13,
         289: 90,
         291: 12,
         290: 101,
         288: 12,
         293: 46,
         286: 3,
         287: 5,
         295: 10,
         294: 10,
         299: 5,
         296: 9,
         298: 42,
         300: 26,
         297: 12,
         303: 196,
         302: 9,
         304: 16,
         301: 22,
         305: 23,
         338: 34,
         315: 4,
         319: 58,
         322: 22,
         308: 1,
         353: 2,
         347: 11,
         324: 36,
         327: 14,
         346: 254,
         306: 346,
         313: 11,
         318: 4,
         334: 2,
         321: 22,
         337: 1,
         352: 9,
         330: 7,
         336: 3,
         350: 5,
         335: 1,
         333: 3,
         345: 1,
         349: 1,
         339: 2,
         309: 1,
         331: 73,
         342: 4,
         316: 1,
         340: 1,
         310: 5,
         343: 2,
         307: 14,
         341: 4,
         348: 1,
         328: 65,
         332: 18,
         326: 1,
         314: 1,
         311: 5,
         320: 1,
         354: 5,
         351: 52,
         325: 1,
         312: 1,
         317: 11,
         329: 25,
         323: 7,
         344: 2,
         358: 39,
         360: 15,
         357: 74,
         355: 121,
         356: 5,
         359: 1,
         362: 1,
         361: 1,
         363: 32,
         364: 4,
         371: 35,
         375: 3,
         368: 7,
         367: 2,
         369: 65,
         373: 1,
         366: 4,
         376: 1,
         374: 3,
         372: 19,
         370: 138,
         365: 14,
         385: 100,
         378: 21,
         383: 31,
         380: 11,
         393: 80,
         391: 7,
         379: 15,
         377: 36,
         382: 21,
         386: 89,
         392: 54,
         389: 4,
         388: 1,
         381: 1,
         390: 3,
         394: 48,
         384: 7,
         387: 20,
         400: 3,
         398: 12,
         399: 5,
         397: 3,
         401: 65,
         395: 4,
         396: 9,
         402: 20,
         403: 178,
         405: 35,
         407: 2,
         404: 26,
         406: 15,
         409: 35,
         416: 41,
         415: 18,
         408: 2,
         410: 32,
         411: 35,
         413: 3,
         412: 10,
         414: 28,
         417: 30,
         418: 3,
         421: 20,
         420: 75,
         419: 36,
         423: 46,
         422: 21,
         424: 18,
         426: 20,
         425: 30,
         433: 16,
         432: 2,
         434: 15,
         428: 4,
         431: 23,
         429: 16,
         427: 15,
         430: 13,
         440: 12,
         441: 14,
         444: 1,
         439: 8,
         443: 82,
         438: 41,
         442: 37,
         437: 2,
         435: 3,
         436: 10,
         446: 85,
         445: 26,
         448: 2,
         449: 19,
         447: 4,
         453: 19,
         451: 33,
         450: 5,
         452: 5,
         454: 15,
         462: 3,
         464: 4,
         457: 5,
         459: 2,
         456: 19,
         463: 13,
         460: 23,
         466: 91,
         455: 17,
         465: 12,
         461: 17,
         458: 1,
         467: 68,
         469: 10,
         468: 16,
         475: 3,
         470: 16,
         472: 2,
         474: 30,
         473: 20,
         471: 48,
         481: 1,
         477: 27,
         476: 11,
         479: 2,
         480: 3,
         482: 2,
         478: 1,
         483: 28,
         484: 1,
         485: 19,
         491: 8,
         490: 8,
         487: 2,
         486: 2,
         493: 4,
         488: 2,
         489: 4,
         492: 4,
         495: 3,
         494: 14,
         496: 1,
         497: 11,
         499: 24,
         498: 12,
         500: 2,
         501: 11,
         502: 106,
         509: 5,
         508: 3,
         516: 2,
         515: 15,
         514: 31,
         513: 3,
         510: 8,
         507: 54,
         503: 32,
         506: 1,
         512: 1,
         505: 1,
         504: 4,
         511: 4,
         517: 5,
         520: 30,
         518: 4,
         519: 1,
         521: 1,
         522: 1,
         524: 6,
         523: 56,
         525: 5,
         526: 3,
         532: 12,
         535: 20,
         529: 22,
         527: 107,
         530: 17,
         534: 23,
         531: 6,
         533: 25,
         528: 1,
         536: 44,
         537: 49,
         539: 11,
         538: 1,
         540: 21,
         542: 9,
         541: 1,
         543: 6,
         544: 5,
         553: 13,
         552: 18,
         556: 23,
         557: 6,
         549: 34,
         551: 3,
         554: 1,
         555: 8,
         547: 1,
         550: 16,
         546: 3,
         558: 33,
         545: 20,
         548: 188,
         559: 21,
         561: 5,
         560: 10,
         567: 12,
         566: 12,
         568: 12,
         570: 3,
         562: 44,
         563: 1,
         565: 10,
         569: 19,
         564: 64,
         571: 16,
         572: 3,
         573: 3,
         575: 8,
         574: 24,
         580: 44,
         586: 107,
         582: 19,
         585: 2,
         577: 3,
         584: 3,
         581: 11,
         578: 1,
         579: 10,
         583: 36,
         587: 29,
         576: 14,
         590: 90,
         591: 2,
         588: 15,
         589: 13,
         592: 14,
         597: 5,
         602: 26,
         601: 36,
         599: 16,
         595: 37,
         596: 6,
         594: 7,
         600: 3,
         598: 1,
         593: 4,
         616: 3,
         605: 1,
         611: 1,
         622: 1,
         610: 40,
         606: 1,
         603: 22,
         614: 52,
         617: 3,
         618: 24,
         620: 2,
         612: 1,
         607: 9,
         613: 7,
         615: 10,
         609: 1,
         608: 30,
         604: 3,
         619: 4,
         621: 9,
         624: 1,
         623: 3,
         625: 13,
         627: 13,
         626: 3,
         628: 6,
         629: 3,
         630: 2,
         636: 5,
         638: 3,
         631: 7,
         635: 2,
         632: 72,
         633: 2,
         637: 118,
         634: 3,
         639: 26,
         641: 3,
         642: 6,
         643: 1,
         644: 38,
         640: 47,
         645: 4,
         646: 1,
         648: 6,
         647: 3,
         650: 12,
         649: 152,
         653: 2,
         651: 11,
         652: 10,
         654: 62,
         655: 95,
         657: 21,
         656: 4,
         658: 17,
         662: 14,
         661: 4,
         659: 10,
         663: 9,
         660: 11,
         664: 6,
         666: 1,
         667: 27,
         669: 7,
         670: 2,
         668: 2,
         665: 9,
         671: 10,
         672: 53,
         677: 111,
         676: 49,
         679: 44,
         673: 7,
         674: 8,
         678: 2,
         675: 3,
         681: 25,
         680: 20,
         682: 10,
         683: 5,
         684: 5,
         690: 5,
         685: 23,
         687: 17,
         689: 2,
         691: 4,
         688: 1,
         686: 21,
         695: 31,
         694: 36,
         692: 3,
         693: 20,
         697: 5,
         696: 4,
         706: 5,
         700: 4,
         708: 1,
         698: 1,
         704: 1,
         705: 20,
         701: 1,
         699: 5,
         702: 8,
         703: 1,
         707: 29,
         710: 21,
         711: 13,
         709: 1,
         712: 5,
         714: 9,
         713: 67,
         716: 2,
         717: 6,
         715: 7,
         718: 27,
         721: 6,
         719: 2,
         720: 21,
         722: 3,
         724: 25,
         723: 18,
         726: 11,
         727: 26,
         725: 13,
         728: 65,
         732: 1,
         735: 29,
         731: 2,
         743: 2,
         729: 42,
         730: 1,
         747: 2,
         733: 3,
         734: 6,
                 736: 1,
                 749: 20,
                 746: 4,
                 739: 25,
                 748: 2,
                 740: 3,
                 745: 19,
                 741: 13,
                 737: 1,
                 744: 31,
                 742: 3,
                 738: 2,
                 750: 10,
                 751: 6,
                 752: 14,
                 753: 7,
                 754: 19,
                 756: 14,
                 755: 13,
                 757: 4,
                 758: 10,
                 760: 13,
                 759: 11,
                 761: 1,
                 762: 15,
                 765: 11,
                 764: 2,
                 763: 9,
                 766: 7,
                 767: 2,
                 768: 21,
                 772: 2,
                 771: 23,
                 770: 23,
                 769: 2,
                 773: 28,
                 792: 1,
                 804: 1,
                 775: 1,
                 807: 2,
                 788: 2,
                 797: 1,
                 793: 21,
                 787: 3,
                 785: 1,
                 798: 4,
                 805: 1,
                 783: 11,
                 791: 6,
                 776: 1,
                 794: 3,
                 774: 19,
                 808: 1,
                 777: 1,
                 806: 29,
                 796: 1,
                 790: 2,
                 781: 1,
                 789: 21,
                 810: 3,
                 784: 1,
                 800: 1,
                 803: 2,
                 782: 1,
                 786: 7,
                 780: 2,
                 802: 1,
                 801: 35,
                 799: 11,
                 779: 3,
                 809: 5,
                 795: 6,
                 778: 1,
                 811: 18,
                 815: 10,
                 812: 13,
                 814: 6,
                 816: 2,
                 818: 37,
                 813: 2,
                 817: 16,
                 824: 37,
                 819: 3,
                 823: 3,
                 821: 4,
                 822: 11,
                 820: 23,
                 826: 3,
                 833: 10,
                 827: 10,
                 825: 19,
                 832: 21,
                 830: 4,
                 828: 10,
                 831: 9,
                 829: 8,
                 835: 25,
                 838: 34,
                 839: 3,
                 836: 3,
                 837: 10,
                 834: 5,
                 841: 10,
                 840: 10,
                 842: 2,
                 843: 11,
                 847: 5,
                 846: 3,
                 845: 2,
                 844: 1,
                 852: 7,
                 850: 7,
                 849: 38,
                 851: 1,
                 848: 11,
                 855: 7,
                 856: 1,
                 853: 3,
                 854: 12,
                 858: 7,
                 860: 44,
                 861: 6,
                 859: 1,
                 857: 3,
                 862: 3,
                 864: 1,
                 863: 6,
                 867: 24,
                 865: 21,
                 866: 3,
                 868: 5,
                 869: 2,
                 871: 13,
                 872: 7,
                 870: 1,
                 873: 16,
                 875: 7,
                 874: 40,
                 877: 13,
                 878: 6,
                 876: 3,
                 879: 1,
                 883: 37,
                 881: 5,
                 880: 5,
                 882: 3,
                 885: 39,
                 888: 3,
                 884: 1,
                 887: 3,
                 886: 11,
                 889: 6,
                 890: 12,
                 894: 4,
                 893: 6,
                 891: 1,
                 892: 5,
                 895: 11,
                 898: 12,
                 896: 2,
                 897: 4,
                 907: 3,
                 903: 7,
                 904: 19,
         908: 1,
         902: 6,
         906: 1,
         899: 1,
         905: 5,
         909: 10,
         901: 2,
         900: 12,
         911: 14,
         910: 24,
         912: 41,
         913: 2,
         914: 42,
         915: 3,
         918: 28,
         917: 1,
         916: 4,
         920: 6,
         919: 25,
         934: 15,
         923: 2,
         924: 10,
         930: 8,
         922: 9,
         931: 5,
         932: 1,
         928: 60,
         933: 1,
         927: 4,
         925: 2,
         935: 22,
         921: 5,
         929: 1,
         926: 1,
         936: 5,
         938: 21,
         937: 9,
         939: 18,
         940: 12,
         941: 11,
         942: 14,
         943: 7,
         946: 2,
         944: 3,
         945: 6,
         948: 2,
         947: 6,
         950: 6,
         951: 14,
         949: 7,
         957: 12,
         952: 2,
         953: 3,
         954: 6,
         956: 4,
         955: 5,
         958: 100,
         959: 5,
         960: 25,
         961: 25,
         964: 6,
         962: 6,
         963: 4,
         967: 1,
         970: 26,
         984: 9,
         983: 6,
         978: 19,
         972: 14,
         969: 11,
         976: 3,
         977: 3,
         981: 1,
         979: 3,
         974: 4,
         968: 2,
         971: 3,
         965: 1,
         973: 1,
         975: 2,
         966: 7,
         980: 1,
         982: 3,
         987: 7,
         985: 5,
         986: 5,
         989: 2,
         991: 4,
         990: 2,
         988: 2,
         992: 9,
         994: 1,
         993: 3,
         995: 2,
         996: 2,
         998: 1,
         997: 6,
         1002: 5,
         ...}
```

In [ ]:
```python
# in how many documents a token appears
dictionary.dfs
```

```
Out[ ]: {9: 169,
         3: 372,
         7: 616,
         1: 157,
         10: 559,
         8: 23,
         6: 128,
         0: 153,
         2: 43,
         5: 192,
         4: 49,
         14: 180,
         12: 71,
         16: 111,
         20: 186,
         13: 41,
         15: 265,
         17: 37,
         11: 336,
         19: 1880,
         18: 1,
         22: 320,
         24: 341,
         40: 499,
         31: 1,
         21: 12,
         26: 1,
         44: 1065,
         38: 20,
         29: 69,
         23: 1318,
         27: 1,
         47: 319,
         36: 165,
         35: 826,
         32: 10,
         41: 65,
         28: 187,
         33: 6,
         43: 1,
         30: 623,
         45: 2,
         46: 164,
         34: 71,
         37: 83,
         25: 82,
         39: 45,
         42: 13,
         65: 183,
         51: 6,
         64: 155,
         56: 49,
         66: 5,
         50: 374,
         52: 10,
         59: 120,
         48: 24,
         54: 582,
         61: 2,
         63: 33,
         55: 7,
         62: 181,
         58: 158,
         60: 56,
         49: 69,
         57: 66,
         53: 68,
         67: 40,
         70: 542,
         68: 46,
         69: 47,
         71: 509,
         74: 52,
         76: 325,
         75: 77,
         73: 291,
         72: 126,
         77: 17,
         85: 95,
         82: 21,
         80: 3,
         78: 349,
         79: 138,
         81: 48,
         83: 59,
         84: 41,
         93: 1,
         95: 213,
         86: 86,
         87: 19,
         91: 130,
         96: 137,
         94: 2,
         88: 112,
         92: 281,
         90: 106,
         97: 29,
         89: 11,
         108: 122,
         106: 91,
         98: 242,
         102: 27,
         105: 3,
         107: 98,
         103: 30,
         104: 10,
         99: 29,
         100: 17,
         101: 13,
         109: 15,
         117: 342,
         110: 4,
         116: 34,
         114: 40,
         111: 30,
         112: 95,
         115: 261,
         118: 67,
         119: 85,
         120: 31,
         113: 2,
         122: 3,
         125: 13,
         130: 3,
         132: 128,
         129: 9,
         121: 5,
         127: 25,
         126: 1,
         131: 2,
         128: 69,
         123: 17,
         124: 142,
         133: 6,
         136: 20,
         135: 2,
         134: 73,
         147: 30,
         146: 43,
         143: 8,
         137: 27,
         139: 11,
         145: 75,
         144: 37,
         140: 6,
         138: 8,
         142: 126,
         141: 370,
         150: 8,
         148: 8,
         152: 44,
         151: 19,
         149: 11,
         153: 17,
         161: 11,
         154: 21,
         158: 9,
         157: 326,
         162: 8,
         155: 111,
         159: 76,
         156: 5,
         160: 52,
         163: 6,
                     165: 41,
                     164: 201,
                     166: 14,
                     168: 64,
                     167: 10,
                     169: 31,
                     170: 51,
                     171: 70,
                     186: 1,
                     180: 75,
                     175: 117,
                     173: 22,
                     178: 22,
                     190: 4,
                     174: 112,
                     176: 17,
                     184: 4,
                     196: 141,
                     189: 3,
                     187: 1,
                     182: 23,
                     192: 86,
                     194: 6,
                     185: 1,
                     193: 65,
                     188: 11,
                     177: 10,
                     191: 27,
                     181: 75,
                     195: 23,
                     179: 45,
                     172: 32,
                     183: 6,
                     199: 29,
                     197: 13,
                     198: 4,
                     205: 103,
                     204: 5,
                     201: 96,
                     200: 9,
                     202: 177,
                     203: 112,
                     216: 41,
                     206: 110,
                     213: 5,
                     226: 7,
                     225: 19,
                     214: 106,
                     210: 32,
                     220: 26,
                     217: 10,
                     212: 17,
                     209: 1,
                     208: 39,
                     219: 10,
                     207: 92,
                     224: 7,
                     218: 85,
                     223: 48,
                     215: 3,
                     211: 6,
                     221: 7,
                     222: 13,
                     252: 142,
                     229: 69,
                     250: 27,
                     238: 208,
                     228: 21,
                     243: 69,
                     240: 84,
                     241: 4,
                     251: 205,
                     242: 23,
                     227: 13,
                     248: 27,
                     236: 24,
                     234: 7,
                     239: 112,
                     237: 2,
                     233: 3,
                     249: 4,
                     247: 2,
         232: 1,
         230: 23,
         235: 5,
         244: 13,
         246: 15,
         245: 4,
         231: 11,
         256: 3,
         253: 2,
         257: 22,
         259: 29,
         254: 2,
         258: 74,
         255: 49,
         270: 23,
         261: 10,
         271: 5,
         276: 3,
         285: 4,
         281: 2,
         266: 4,
         273: 6,
         265: 8,
         279: 19,
         272: 2,
         268: 1,
         263: 1,
         280: 1,
         264: 1,
         275: 1,
         260: 1,
         277: 1,
         284: 1,
         283: 1,
         269: 13,
         262: 1,
         282: 1,
         278: 54,
         267: 19,
         274: 82,
         292: 13,
         289: 87,
         291: 12,
         290: 99,
         288: 11,
         293: 46,
         286: 3,
         287: 5,
         295: 10,
         294: 9,
         299: 5,
         296: 9,
         298: 42,
         300: 26,
         297: 12,
         303: 173,
         302: 9,
         304: 16,
         301: 22,
         305: 23,
         338: 33,
         315: 4,
         319: 57,
         322: 20,
         308: 1,
         353: 2,
         347: 11,
         324: 34,
         327: 14,
         346: 227,
         306: 319,
         313: 9,
         318: 4,
         334: 2,
         321: 22,
         337: 1,
         352: 9,
         330: 7,
         336: 3,
         350: 4,
         335: 1,
         333: 3,
             345: 1,
             349: 1,
             339: 2,
             309: 1,
             331: 72,
             342: 4,
             316: 1,
             340: 1,
             310: 5,
             343: 2,
             307: 14,
             341: 4,
             348: 1,
             328: 62,
             332: 18,
             326: 1,
             314: 1,
             311: 5,
             320: 1,
             354: 5,
             351: 52,
             325: 1,
             312: 1,
             317: 11,
             329: 24,
             323: 7,
             344: 2,
             358: 34,
             360: 15,
             357: 63,
             355: 114,
             356: 5,
             359: 1,
             362: 1,
             361: 1,
             363: 32,
             364: 4,
             371: 35,
             375: 3,
             368: 7,
             367: 2,
             369: 61,
             373: 1,
             366: 4,
             376: 1,
             374: 3,
             372: 19,
             370: 136,
             365: 14,
             385: 94,
             378: 21,
             383: 31,
             380: 11,
             393: 78,
             391: 7,
             379: 15,
             377: 35,
             382: 21,
             386: 67,
             392: 53,
             389: 4,
             388: 1,
             381: 1,
             390: 3,
             394: 48,
             384: 7,
             387: 20,
             400: 3,
             398: 11,
             399: 5,
             397: 3,
             401: 59,
             395: 4,
             396: 9,
             402: 20,
             403: 178,
             405: 33,
             407: 2,
             404: 26,
             406: 15,
             409: 34,
             416: 41,
             415: 18,
             408: 2,
             410: 32,
             411: 35,
             413: 3,
             412: 10,
             414: 28,
             417: 29,
             418: 3,
             421: 20,
             420: 71,
             419: 36,
             423: 44,
             422: 21,
             424: 18,
             426: 20,
             425: 30,
             433: 16,
             432: 2,
             434: 15,
             428: 4,
             431: 22,
             429: 16,
             427: 15,
             430: 13,
             440: 12,
             441: 14,
             444: 1,
             439: 8,
             443: 82,
             438: 39,
             442: 37,
             437: 2,
             435: 3,
             436: 9,
             446: 82,
             445: 26,
             448: 2,
             449: 19,
             447: 4,
             453: 19,
             451: 33,
             450: 5,
             452: 5,
             454: 15,
             462: 3,
             464: 4,
             457: 5,
             459: 2,
             456: 18,
             463: 12,
             460: 22,
             466: 87,
             455: 17,
             465: 12,
             461: 17,
             458: 1,
             467: 68,
             469: 10,
             468: 16,
             475: 3,
             470: 15,
             472: 2,
             474: 29,
             473: 20,
             471: 47,
             481: 1,
             477: 27,
             476: 10,
             479: 2,
             480: 3,
             482: 2,
             478: 1,
             483: 28,
             484: 1,
             485: 19,
             491: 8,
             490: 6,
             487: 2,
             486: 2,
             493: 4,
             488: 2,
         489: 4,
         492: 4,
         495: 3,
         494: 12,
         496: 1,
         497: 11,
         499: 24,
         498: 12,
         500: 2,
         501: 11,
         502: 106,
         509: 5,
         508: 3,
         516: 2,
         515: 14,
         514: 29,
         513: 3,
         510: 8,
         507: 52,
         503: 32,
         506: 1,
         512: 1,
         505: 1,
         504: 4,
         511: 4,
         517: 5,
         520: 30,
         518: 4,
         519: 1,
         521: 1,
         522: 1,
         524: 6,
         523: 56,
         525: 5,
         526: 3,
         532: 12,
         535: 19,
         529: 22,
         527: 106,
         530: 17,
         534: 23,
         531: 6,
         533: 25,
         528: 1,
         536: 44,
         537: 48,
         539: 11,
         538: 1,
         540: 17,
         542: 9,
         541: 1,
         543: 6,
         544: 5,
         553: 13,
         552: 18,
         556: 23,
         557: 6,
         549: 34,
         551: 3,
         554: 1,
         555: 8,
         547: 1,
         550: 15,
         546: 3,
         558: 31,
         545: 20,
         548: 160,
         559: 20,
         561: 5,
         560: 10,
         567: 12,
         566: 12,
         568: 12,
         570: 3,
         562: 44,
         563: 1,
         565: 10,
         569: 19,
         564: 64,
         571: 16,
         572: 3,
         573: 3,
         575: 8,
         574: 24,
         580: 43,
         586: 104,
         582: 19,
         585: 2,
         577: 3,
         584: 3,
         581: 10,
         578: 1,
         579: 9,
         583: 34,
         587: 29,
         576: 14,
         590: 86,
         591: 2,
         588: 15,
         589: 13,
         592: 14,
         597: 5,
         602: 26,
         601: 35,
         599: 16,
         595: 35,
         596: 6,
         594: 7,
         600: 3,
         598: 1,
         593: 4,
         616: 3,
         605: 1,
         611: 1,
         622: 1,
         610: 38,
         606: 1,
         603: 21,
         614: 52,
         617: 3,
         618: 24,
         620: 2,
         612: 1,
         607: 7,
         613: 7,
         615: 10,
         609: 1,
         608: 29,
         604: 3,
         619: 4,
         621: 8,
         624: 1,
         623: 3,
         625: 13,
         627: 13,
         626: 3,
         628: 6,
         629: 3,
         630: 2,
         636: 5,
         638: 3,
         631: 7,
         635: 2,
         632: 70,
         633: 2,
         637: 108,
         634: 3,
         639: 24,
         641: 3,
         642: 6,
         643: 1,
         644: 37,
         640: 47,
         645: 4,
         646: 1,
         648: 6,
         647: 3,
         650: 12,
         649: 140,
         653: 2,
         651: 11,
         652: 9,
         654: 59,
         655: 95,
         657: 21,
         656: 4,
         658: 17,
         662: 13,
         661: 4,
         659: 10,
         663: 8,
         660: 11,
         664: 6,
         666: 1,
         667: 23,
         669: 7,
         670: 2,
         668: 2,
         665: 9,
         671: 10,
         672: 53,
         677: 111,
         676: 47,
         679: 43,
         673: 7,
         674: 8,
         678: 2,
         675: 3,
         681: 25,
         680: 20,
         682: 10,
         683: 5,
         684: 5,
         690: 5,
         685: 23,
         687: 16,
         689: 2,
         691: 4,
         688: 1,
         686: 20,
         695: 31,
         694: 35,
         692: 3,
         693: 19,
         697: 5,
         696: 4,
         706: 5,
         700: 4,
         708: 1,
         698: 1,
         704: 1,
         705: 19,
         701: 1,
         699: 5,
         702: 8,
         703: 1,
         707: 28,
         710: 21,
         711: 13,
         709: 1,
         712: 5,
         714: 9,
         713: 67,
         716: 2,
         717: 6,
         715: 7,
         718: 27,
         721: 5,
         719: 2,
         720: 20,
         722: 3,
         724: 24,
         723: 18,
         726: 11,
         727: 26,
         725: 13,
         728: 64,
         732: 1,
         735: 28,
         731: 2,
         743: 2,
         729: 41,
         730: 1,
         747: 2,
         733: 3,
         734: 5,
         736: 1,
         749: 20,
         746: 4,
         739: 24,
         748: 2,
         740: 3,
         745: 19,
         741: 12,
         737: 1,
         744: 31,
         742: 3,
         738: 2,
         750: 10,
         751: 6,
         752: 13,
         753: 7,
         754: 19,
         756: 14,
         755: 13,
         757: 4,
         758: 10,
         760: 13,
         759: 11,
         761: 1,
         762: 15,
         765: 11,
         764: 2,
         763: 8,
         766: 7,
         767: 2,
         768: 21,
         772: 2,
         771: 22,
         770: 22,
         769: 2,
         773: 26,
         792: 1,
         804: 1,
         775: 1,
         807: 1,
         788: 1,
         797: 1,
         793: 21,
         787: 3,
         785: 1,
         798: 4,
         805: 1,
         783: 11,
         791: 6,
         776: 1,
         794: 3,
         774: 14,
         808: 1,
         777: 1,
         806: 29,
         796: 1,
         790: 2,
         781: 1,
         789: 20,
         810: 3,
         784: 1,
         800: 1,
         803: 2,
         782: 1,
         786: 7,
         780: 2,
         802: 1,
         801: 34,
         799: 11,
         779: 3,
         809: 5,
         795: 6,
         778: 1,
         811: 18,
         815: 10,
         812: 13,
         814: 6,
         816: 2,
         818: 34,
         813: 2,
         817: 16,
         824: 35,
         819: 3,
         823: 3,
         821: 4,
         822: 11,
         820: 23,
         826: 3,
         833: 8,
         827: 10,
         825: 19,
         832: 20,
         830: 4,
         828: 10,
         831: 9,
         829: 8,
         835: 25,
         838: 33,
         839: 3,
         836: 3,
         837: 10,
         834: 5,
         841: 10,
         840: 10,
         842: 2,
         843: 11,
         847: 5,
         846: 3,
         845: 2,
         844: 1,
         852: 7,
         850: 7,
         849: 38,
         851: 1,
         848: 11,
         855: 7,
         856: 1,
         853: 3,
         854: 12,
         858: 7,
         860: 43,
         861: 6,
         859: 1,
         857: 3,
         862: 3,
         864: 1,
         863: 5,
         867: 24,
         865: 20,
         866: 3,
         868: 5,
         869: 2,
         871: 13,
         872: 7,
         870: 1,
         873: 16,
         875: 6,
         874: 40,
         877: 13,
         878: 6,
         876: 3,
         879: 1,
         883: 35,
         881: 5,
         880: 5,
         882: 3,
         885: 34,
         888: 3,
         884: 1,
         887: 3,
         886: 10,
         889: 5,
         890: 12,
         894: 4,
         893: 6,
         891: 1,
         892: 5,
         895: 11,
         898: 12,
         896: 2,
         897: 4,
         907: 3,
         903: 6,
         904: 18,
         908: 1,
         902: 6,
         906: 1,
         899: 1,
         905: 5,
         909: 9,
         901: 2,
         900: 12,
         911: 14,
         910: 24,
         912: 41,
         913: 2,
         914: 42,
         915: 3,
         918: 28,
         917: 1,
         916: 4,
         920: 6,
         919: 25,
         934: 15,
         923: 2,
         924: 10,
         930: 8,
         922: 8,
         931: 5,
         932: 1,
         928: 60,
         933: 1,
         927: 4,
         925: 2,
         935: 22,
         921: 5,
         929: 1,
         926: 1,
         936: 5,
         938: 21,
         937: 9,
         939: 18,
         940: 12,
         941: 11,
         942: 14,
         943: 7,
         946: 2,
         944: 3,
         945: 6,
         948: 2,
         947: 6,
         950: 6,
         951: 14,
         949: 7,
         957: 12,
         952: 2,
         953: 3,
         954: 6,
         956: 3,
         955: 5,
         958: 97,
         959: 5,
         960: 25,
         961: 24,
         964: 5,
         962: 6,
         963: 4,
         967: 1,
         970: 26,
         984: 9,
         983: 6,
         978: 19,
         972: 14,
         969: 11,
         976: 2,
         977: 2,
         981: 1,
         979: 2,
         974: 4,
         968: 2,
         971: 3,
         965: 1,
         973: 1,
         975: 2,
         966: 7,
         980: 1,
         982: 3,
         987: 7,
         985: 5,
         986: 5,
         989: 2,
         991: 2,
         990: 2,
         988: 2,
         992: 9,
         994: 1,
         993: 2,
         995: 1,
         996: 1,
         998: 1,
         997: 6,
         1002: 5,
         ...}
```

In [ ]:
```python
'''
We can convert a list of tokens into a list of their corresponding indices in
the vocabulary.
'''


sentence = 'follow the white rabbit'
new_tokens = simple_preprocess(sentence)
ids = dictionary.doc2idx(new_tokens)
print(new_tokens)
print(ids)
```

```
['follow', 'the', 'white', 'rabbit']
[156, 19, 818, 885]
```

Despite the size of our vocabulary, we need to think that there will always be a word that isn't include...

Therefore, we use special token:

- `[UNK]` for unknown
- `[PAD]` to pad the short sentences

We can add them to our voc.

In [ ]:
```python
special_tokens = {'[PAD]': 0, '[UNK]': 1}
dictionary.patch_with_special_tokens(special_tokens)
```

Again, we could also be interested in removing rare terms as well as 'bad words'.

Gensim has a couple of methods that can help on that:

- `filter_extremes()`
- `filter_tokens()`

In [ ]:
```python
def get_rare_ids(dictionary, min_freq):
    rare_ids = [t[0] for t in dictionary.cfs.items() if t[1] < min_freq]
    return rare_ids
```

In [ ]:
```
'''
The code below wraps everything togheter:
it takes a list of sentences, generates the corresponding vocabulary
and it saves it
'''

def make_vocab(sentences, folder=None, special_tokens=None, vocab_size=None, min_freq=None):
    if folder is not None:
        if not os.path.exists(folder):
            os.mkdir(folder)

    # tokenizes the sentences and create a Dictionary
    tokens = [simple_preprocess(sent) for sent in sentences]
    dictionary = corpora.Dictionary(tokens)
    # keeps only the most frequent words (vocab size)
    if vocab_size is not None:
        dictionary.filter_extremes(keep_n=vocab_size)
    # removes rare words (in case the vocab size still
    # includes words with low frequency)
    if min_freq is not None:
        rare_tokens = get_rare_ids(dictionary, min_freq)
        dictionary.filter_tokens(bad_ids=rare_tokens)
    # gets the whole list of tokens and frequencies
    items = dictionary.cfs.items()
    # sorts the tokens in descending order
    words = [dictionary[t[0]] for t in sorted(dictionary.cfs.items(), key=lambda t: -t[1])]
    # prepends special tokens, if any
    if special_tokens is not None:
        to_add = []
        for special_token in special_tokens:
            if special_token not in words:
                to_add.append(special_token)
        words = to_add + words

    with open(os.path.join(folder, 'vocab.txt'), 'w') as f:
        for word in words:
            f.write(f'{word}\n')
```

In [ ]:
```
make_vocab(train_dataset['sentence'], 'our_vocab/', special_tokens=['[PAD]', '[UNK]', '[SEP]', '[CLS]', '[MAS
K]'], min_freq=2)
```

HugginFace's Tokenizer

Since we are going to use BERT, we will use the corresponding pre-trained tokenizer.

It standardize, in some sense, the input for BERT. It has the same information as gensim, i.e., the mapping between tokens and their id, but it includes many more information.

In [ ]:
```
'''
It takes a vocabulary as input
'''

from transformers import BertTokenizer

tokenizer = BertTokenizer('our_vocab/vocab.txt')
```

In [ ]:
```
new_sentence = 'follow the white rabbit neo'
new_tokens = tokenizer.tokenize(new_sentence)
new_tokens
```

Out[ ]:
```
['follow', 'the', 'white', 'rabbit', '[UNK]']
```

In [ ]:
```
new_ids = tokenizer.convert_tokens_to_ids(new_tokens)
new_ids
```

Out[ ]:
```
[979, 5, 207, 200, 1]
```

In [ ]:
```
new_ids = tokenizer.encode(new_sentence)
new_ids
```

Out[ ]:
```
[3, 979, 5, 207, 200, 1, 2]
```

In [ ]:
```
tokenizer.convert_ids_to_tokens(new_ids)
```

Out[ ]:
```
['[CLS]', 'follow', 'the', 'white', 'rabbit', '[UNK]', '[SEP]']
```

In Transformer-based models, particularly BERT (Bidirectional Encoder Representations from Transformers), the [CLS] token stands for Classification token.

Purpose of the [CLS] Token

Representation: The [CLS] token is a special token added to the beginning of every input sequence in models like BERT. During training, the model learns to treat the embedding of this token as a representation of the entire sequence.

Classification Tasks: For tasks that involve classification (e.g., sentiment analysis, sentence classification, or Next Sentence Prediction), the final hidden state of the [CLS] token (after processing through all transformer layers) is used as a summary of the entire sequence.

General-Purpose Embedding: It’s often considered the “pooled” output, as it attempts to capture information from the whole sequence in a single vector. This embedding can then be passed to a classifier or other downstream layers for decision-making.

In [ ]:
```
tokenizer.encode(new_sentence, add_special_tokens=False)
```

Out[ ]:
```
[979, 5, 207, 200, 1]
```

In [ ]:
```
tokenizer(new_sentence, add_special_tokens=False, return_tensors='pt')
```

Out[ ]:
```
{'input_ids': tensor([[979,   5, 207, 200,   1]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0]]), 'attention_mas
k': tensor([[1, 1, 1, 1, 1]])}
```

In [ ]:
```
sentence1 = 'follow the white rabbit neo'
sentence2 = 'no one can be told what the matrix is'
joined_sentences = tokenizer(sentence1, sentence2)
joined_sentences
```

Out[ ]:
```
{'input_ids': [3, 979, 5, 207, 200, 1, 2, 51, 49, 76, 32, 301, 42, 5, 1, 30, 2], 'token_type_ids': [0, 0, 0,
0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1]}
```

3 level of information:

- input_ids ⇒ OK! we know them
- token_type_ids works as sentence index
- attention_mask that corresponds to our source mask

In [ ]:
```
print(tokenizer.convert_ids_to_tokens(joined_sentences['input_ids']))
```

```
['[CLS]', 'follow', 'the', 'white', 'rabbit', '[UNK]', '[SEP]', 'no', 'one', 'can', 'be', 'told', 'what', 'th
e', '[UNK]', 'is', '[SEP]']
```

In [ ]:
```
separate_sentences = tokenizer([sentence1, sentence2], padding=True)
separate_sentences
```

Out[ ]:
```
{'input_ids': [[3, 979, 5, 207, 200, 1, 2, 0, 0, 0, 0], [3, 51, 49, 76, 32, 301, 42, 5, 1, 30, 2]], 'token_typ
e_ids': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], 'attention_mask': [[1, 1, 1,
1, 1, 1, 1, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]}
```

In [ ]:
```
print(tokenizer.convert_ids_to_tokens(separate_sentences['input_ids'][0]))
print(separate_sentences['attention_mask'][0])
```

```
['[CLS]', 'follow', 'the', 'white', 'rabbit', '[UNK]', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]']
[1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
```

In [ ]:
```
first_sentences = [sentence1, 'another first sentence']
second_sentences = [sentence2, 'a second sentence here']
batch_of_pairs = tokenizer(first_sentences, second_sentences)
first_input = tokenizer.convert_ids_to_tokens(batch_of_pairs['input_ids'][0])
second_input = tokenizer.convert_ids_to_tokens(batch_of_pairs['input_ids'][1])
print(first_input)
print(second_input)
```

```
['[CLS]', 'follow', 'the', 'white', 'rabbit', '[UNK]', '[SEP]', 'no', 'one', 'can', 'be', 'told', 'what', 'th
e', '[UNK]', 'is', '[SEP]']
['[CLS]', 'another', 'first', 'sentence', '[SEP]', '[UNK]', 'second', 'sentence', 'here', '[SEP]']
```

```
In [ ]: tokenized_dataset = tokenizer(dataset['sentence'],
                                      padding=True,
                                      return_tensors='pt',
                                      max_length=50,
                                      truncation=True)
        tokenized_dataset['input_ids']
```

```
Out[ ]: tensor([[   3,   27,    1,  ...,   92, 1070,    2],
                [   3,   24,   10,  ..., 1876,    5,    2],
                [   3,   44,   12,  ...,    0,    0,    0],
                ...,
                [   3,    1,    6,  ...,    0,    0,    0],
                [   3,    6,  129,  ...,    0,    0,    0],
                [   3,    1,    1,  ...,    0,    0,    0]])
```

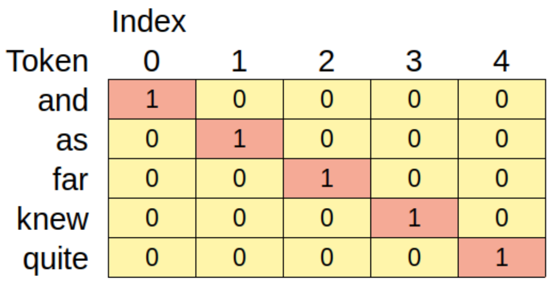
In reality, BERT uses vectors to represent the words, using a big look-up table with the token ids as indeces.

This recalls us... **EMBEDDINGS**, in this case **WORD EMBEDDINGS**, so a representation of each token (word) as a vector, a vector of numbers... The size of the vector is the dimensionality of the embeddings.

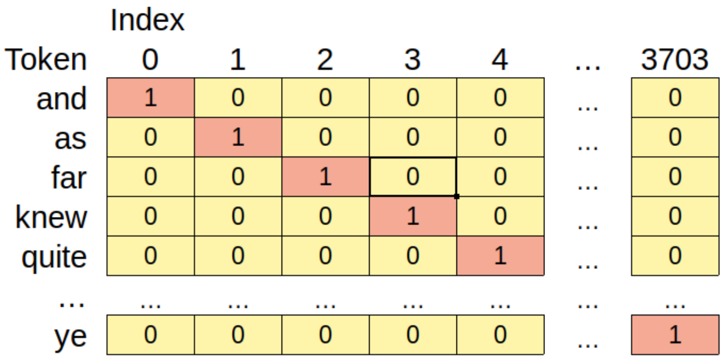
We can build the embeddings or we can learn... then we start a beautiful trip into that.

Before Word Embeddings

One-Hot Encoding (OHE)



	Index				
Token	0	1	2	3	4
and	1	0	0	0	0
as	0	1	0	0	0
far	0	0	1	0	0
knew	0	0	0	1	0
quite	0	0	0	0	1



	Index						
Token	0	1	2	3	4	...	3703
and	1	0	0	0	0	...	0
as	0	1	0	0	0	...	0
far	0	0	1	0	0	...	0
knew	0	0	0	1	0	...	0
quite	0	0	0	0	1	...	0
...	...	...	...	...	...	...	...
ye	0	0	0	0	0	...	1

As you can imagine, a representation like this has some issues:

- large
- sparse (lots of zeros!!!)

Bag of Words (BoW)

Of course, we can do better. We could, for instance, sum up the corresponding OHE vectors, disregarding any underlying structure of relationships between the words.

The result is the counts of the words appearing in the text

```
In [ ]: sentence = 'the white rabbit is a rabbit'
        bow_tokens = simple_preprocess(sentence)
        bow_tokens

Out[ ]: ['the', 'white', 'rabbit', 'is', 'rabbit']
```

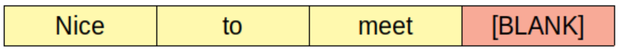
```
In [ ]: bow = dictionary.doc2bow(bow_tokens)
        bow

Out[ ]: [(15, 1), (19, 1), (818, 1), (885, 2)]
```

Also this approach has several limitations:

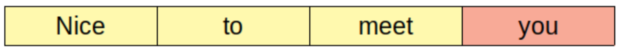
- it represents the frequencies, nothing else
- the representations can be really different when we compute a similarity function (hortogonal in the case of OHE)

Language models, instead, try to explore the structure and the relationships between words

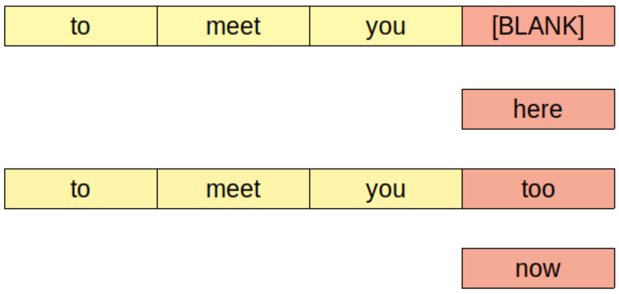
Language Models



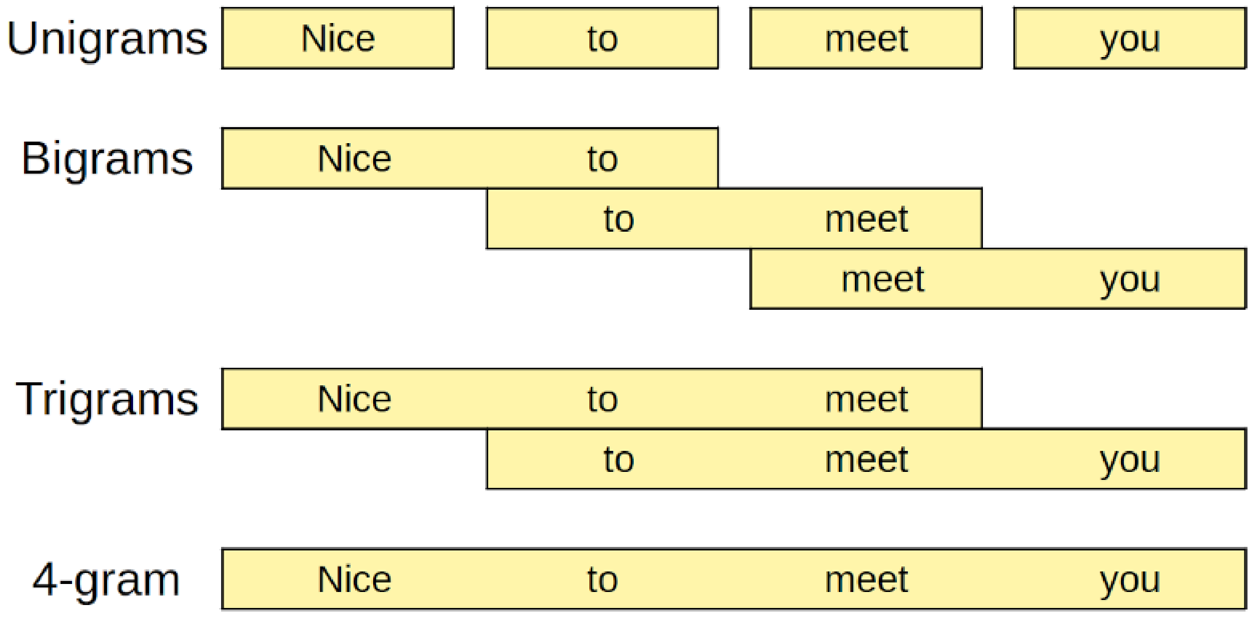
It is easy to fill the gap with YOU



What about this?



N-grams



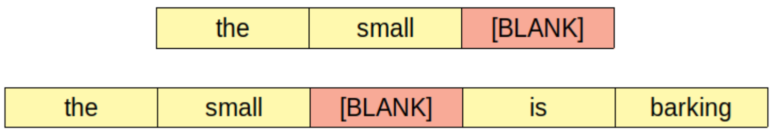
n-grams is base on pure statistics, filling the blanks using the most common sequence that matches the words precceding the blank

With a large n, you might get good predictions, but with many cases with 0 predictions, on the contrary you may encounter many prediction errors.

This is due to the fact that we only **look back**.

Let's try to look ahead too!

Continuous Bag-of-Words (CBoW)



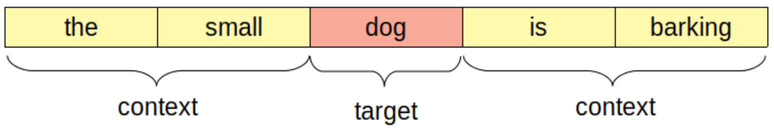
It sums up (or averages) the vectors of the context words and uses it to predit the central word.

The vectors are not one-hot-encoded and have continous values. The vector containing the continous values are called word embeddings.

We can learn these vectors..

Word Embeddings

Word2Vec



The **target** is the central word, therefore we deal with a multiclass classification problem, where the number of classes is the size of the vocabulary.

We use the context words, better their embedding vectors, as input. Therefore becoming a parameter itself of the model.

They are randomly initialized, then as the training progresses, their weights and biases are updated.

How it works

For each pair of the context words, and the corresponding target, the model will average the embeddings of the context and feed the result to a linear layer that will compute one logit for each word in the vocabulary

```
In [ ]: class CBOW(nn.Module):
            def __init__(self, vocab_size, embedding_size):
                super().__init__()
                self.embedding = nn.Embedding(vocab_size, embedding_size)
                self.linear = nn.Linear(embedding_size, vocab_size)

            def forward(self, X):
                embeddings = self.embedding(X)
                bow = embeddings.mean(dim=1)
                logits = self.linear(bow)
                return logits
```

```
In [ ]: torch.manual_seed(42)
        dummy_cbow = CBOW(vocab_size=5, embedding_size=3)
        dummy_cbow.embedding.state_dict()
```

```
Out[ ]: OrderedDict([('weight',
                      tensor([[ 0.3367,  0.1288,  0.2345],
                              [ 0.2303, -1.1229, -0.1863],
                              [ 2.2082, -0.6380,  0.4617],
                              [ 0.2674,  0.5349,  0.8094],
                              [ 1.1103, -1.6898, -0.9890]]))])
```

	Dimensions		
Token	0	1	2
the	0.3367	0.1288	0.2345
small	0.2303	-1.1229	-0.1863
is	2.2082	-0.6380	0.4617
barking	0.2674	0.5349	0.8094
dog	1.1103	-1.6898	-0.9890

nn.Embeddings layer is a lookup table. It may be randomly initialized given the size of the vocabulary and the number of dimensions.

```
In [ ]: # tokens: ['is', 'barking']
        dummy_cbow.embedding(torch.as_tensor([2, 3]))
```

```
Out[ ]: tensor([[ 2.2082, -0.6380,  0.4617],
                [ 0.2674,  0.5349,  0.8094]], grad_fn=<EmbeddingBackward0>)
```

Pretending we have performed a tokenization and we have the indices of our vocab, context and target

```
In [ ]: tiny_vocab = ['the', 'small', 'is', 'barking', 'dog']
        context_words = ['the', 'small', 'is', 'barking']
        target_words = ['dog']
```

```
In [ ]: batch_context = torch.as_tensor([[0, 1, 2, 3]]).long()
        batch_target = torch.as_tensor([4]).long()
```

	Dimensions		
Token	0	1	2
the	0.3367	0.1288	0.2345
small	0.2303	-1.1229	-0.1863
is	2.2082	-0.6380	0.4617
barking	0.2674	0.5349	0.8094
CBOW	0.7607	-0.2743	0.3298

```
In [ ]: cbow_features = dummy_cbow.embedding(batch_context).mean(dim=1)
        cbow_features
```

```
Out[ ]: tensor([[ 0.7606, -0.2743,  0.3298]], grad_fn=<MeanBackward1>)
```

Logits	C	class
0.3542	0	the
0.6937	1	small
-0.2028	2	is
-0.5873	3	barking
0.2099	4	dog

```
In [ ]: logits = dummy_cbow.linear(cbow_features)
        logits
```

```
Out[ ]: tensor([[ 0.3542,  0.6937, -0.2028, -0.5873,  0.2099]],
               grad_fn=<AddmmBackward0>)
```

What is an Embeddings Anyway?

It is a representation, and each dimension of the vector corresponds to an attribute/feature.

For instance, we can describe restaurants with 3 features.

We can represent the values in numbers ranging in the interval [-1,1]

... and we can compute a similarity among them, cosine?

Restaurant	Food	Price	Service
#1	Good	Expensive	Good
#2	Average	Cheap	Bad
#3	Very Good	Expensive	Very Good
#4	Bad	Very Cheap	Average

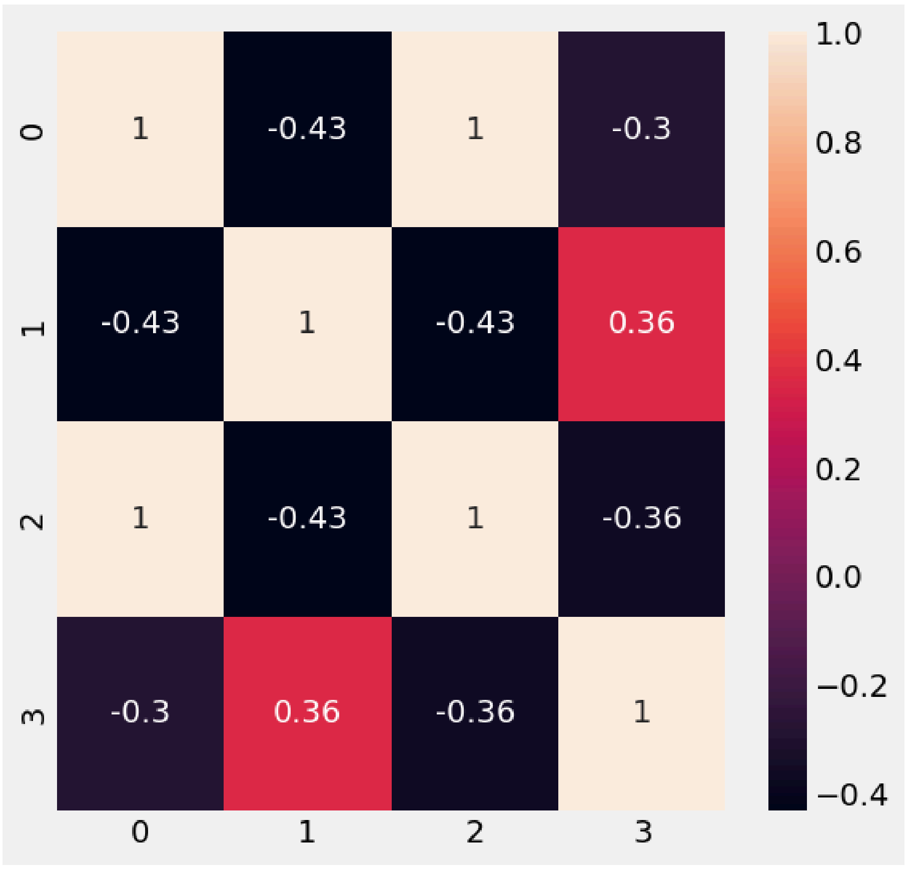
Restaurant	Food	Price	Service
#1	0.70	-0.40	0.70
#2	0.30	0.70	-0.50
#3	0.90	-0.55	0.80
#4	-0.30	0.80	0.34

```
In [ ]: ratings = torch.as_tensor([[.7, -.4, .7],
                                   [.3, .7, -.5],
                                   [.9, -.55, .8],
                                   [-.3, .8, .34]]).float()
        sims = torch.zeros(4, 4)
        for i in range(4):
            for j in range(4):
                sims[i, j] = F.cosine_similarity(ratings[i], ratings[j], dim=0)
        sims
```

```
Out[ ]: tensor([[ 1.0000, -0.4318,  0.9976, -0.2974],
                [-0.4318,  1.0000, -0.4270,  0.3581],
                [ 0.9976, -0.4270,  1.0000, -0.3598],
                [-0.2974,  0.3581, -0.3598,  1.0000]])
```

In [ ]:
```python
import seaborn as sns

fig, ax = plt.subplots(figsize=(6, 6))
sns.heatmap(sims.detach().numpy(), annot=True, ax=ax)
```

Out[ ]:
```
<Axes: >
```



Pre-trained Word2Vec

In general, pretrained-word2vec models and the embeddings correspoding to a word do not have a clear meaning, but we can do fancy stuff with them.

We don't know if the $n-th$ dimension correspond to a particular "behaviour" of the word

To train word2vec that still is a simple model it requires a certain amount of data to train it.

Luckly someone already did it the job for us and for instance gensim contains a variety of pre-trained word embeddings models.

But why different models and therefore different embeddings? Well, using different corpora produces different embeddings, since they might be influenced by the **kind of language** used in the corpora

They might depend on the model used to learn the embeddings, word2vec is one but there are many others.

Global Vectors (GloVe)

GloVe: Global Vectors for Word Representation

It combines skip-gram model with co-occurences statistics at the **global level**

Take it for grant, if you want to know more, you can read the paper and check https://nlp.stanford.edu/projects/glove (https://nlp.stanford.edu/projects/glove)

There are many sizes and shapes, dimensions from 25 to 300 and vocab size between 400,000 and 2,200,000 words.

```
In [ ]: from gensim import downloader

        glove = downloader.load('glove-wiki-gigaword-50')


        len(glove.key_to_index)

Out[ ]: 400000
```

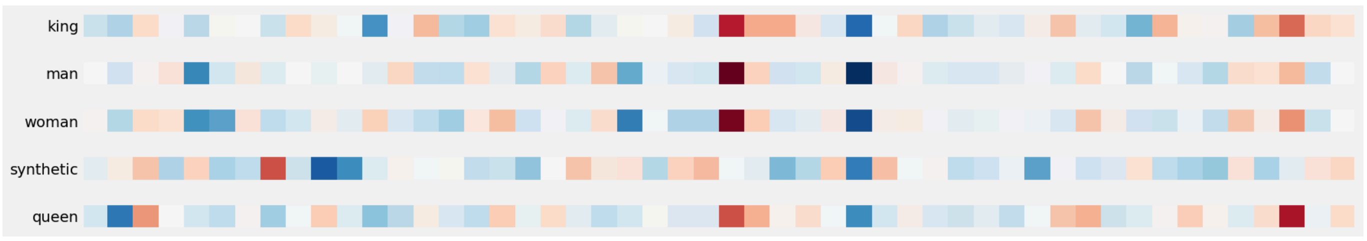
```
In [ ]: glove['alice']

Out[ ]: array([ 0.16386 ,  0.57795 , -0.59197 , -0.32446 ,  0.29762 ,  0.85151 ,
               -0.76695 , -0.20733 ,  0.21491 , -0.51587 , -0.17517 ,  0.94459 ,
                0.12705 , -0.33031 ,  0.75951 ,  0.44449 ,  0.16553 , -0.19235 ,
                0.065533, -0.12394 ,  0.61446 ,  0.89784 ,  0.17413 ,  0.41149 ,
                1.191   , -0.39461 , -0.459   ,  0.022161, -0.50843 , -0.44464 ,
                0.68721 , -0.7167  ,  0.20835 , -0.23437 ,  0.02604 , -0.47993 ,
                0.31873 , -0.29135 ,  0.50273 , -0.55144 , -0.066692,  0.43873 ,
               -0.24293 , -1.0247  ,  0.029375,  0.068499,  0.25451 , -1.9663  ,
                0.26673 ,  0.88486 ], dtype=float32)
```

We don't know the meaning of the dimensions, but we can do math with them

We can define the queen with the following equation: king - man + woman = queen

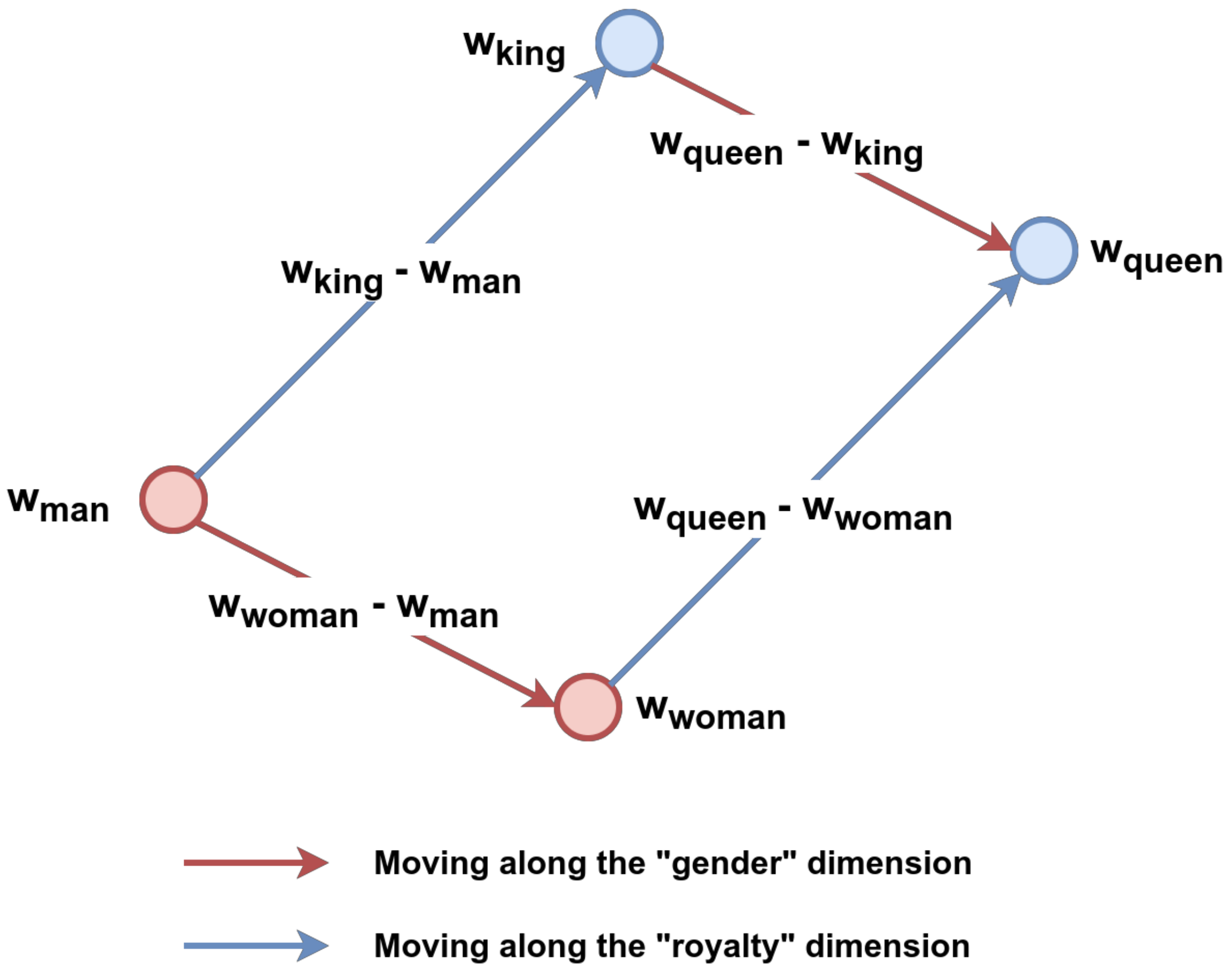
```
In [ ]: synthetic_queen = glove['king'] - glove['man'] + glove['woman']
```

```
In [ ]: fig = plot_word_vectors(glove,
                                ['king', 'man', 'woman', 'synthetic', 'queen'],
                                other={'synthetic': synthetic_queen})

        /content/plots/chapter11.py:23: UserWarning: set_ticklabels() should only be used with a fixed number of tick
        s, i.e. after set_ticks() or using a FixedLocator.
          axs[i].set_yticklabels(['', word, ''])
        /content/plots/chapter11.py:23: UserWarning: set_ticklabels() should only be used with a fixed number of tick
        s, i.e. after set_ticks() or using a FixedLocator.
          axs[i].set_yticklabels(['', word, ''])
        /content/plots/chapter11.py:23: UserWarning: set_ticklabels() should only be used with a fixed number of tick
        s, i.e. after set_ticks() or using a FixedLocator.
          axs[i].set_yticklabels(['', word, ''])
        /content/plots/chapter11.py:23: UserWarning: set_ticklabels() should only be used with a fixed number of tick
        s, i.e. after set_ticks() or using a FixedLocator.
          axs[i].set_yticklabels(['', word, ''])
        /content/plots/chapter11.py:23: UserWarning: set_ticklabels() should only be used with a fixed number of tick
        s, i.e. after set_ticks() or using a FixedLocator.
          axs[i].set_yticklabels(['', word, ''])
```



```
In [ ]: glove.similar_by_vector(synthetic_queen, topn=5)

Out[ ]: [('king', 0.8859834671020508),
         ('queen', 0.8609582185745239),
         ('daughter', 0.7684512734413147),
         ('prince', 0.7640699744224548),
         ('throne', 0.7634970545768738)]
```

It's pretty common that the first results corresponds to the origin of the word embeddings arithmetic, so we can exclude it from the result... and we get queen :-)



$$w_{\text{king}} - w_{\text{man}} \approx w_{\text{queen}} - w_{\text{woman}} \implies w_{\text{king}} - w_{\text{man}} + w_{\text{woman}} \approx w_{\text{queen}}$$

It is nice and it shows that effectively the embeddings are capturing the meaning/semantics of the words.

Using Word Embeddings

Vocabulary Coverage

```
In [ ]: vocab = list(dictionary.token2id.keys())
        len(vocab)

Out[ ]: 3676
```

```
In [ ]: unknown_words = sorted(list(set(vocab).difference(set(glove.key_to_index))))
        ##############################################################
        print(len(unknown_words))
        print(unknown_words[:5])

        39
        ['[PAD]', '[UNK]', 'arrum', 'barrowful', 'beauti']
```

```
In [ ]: unknown_ids = [dictionary.token2id[w] for w in unknown_words if w not in ['[PAD]', '[UNK]']]
        unknown_count = np.sum([dictionary.cfs[idx] for idx in unknown_ids])
        unknown_count, dictionary.num_pos

Out[ ]: (79, 50766)
```

```
In [ ]: def vocab_coverage(gensim_dict, pretrained_wv, special_tokens=('[PAD]', '[UNK]')):
            vocab = list(gensim_dict.token2id.keys())
            unknown_words = sorted(list(set(vocab).difference(set(pretrained_wv.key_to_index))))
            ###########################################################
            unknown_ids = [gensim_dict.token2id[w] for w in unknown_words if w not in special_tokens]
            unknown_count = np.sum([gensim_dict.cfs[idx] for idx in unknown_ids])
            cov = 1 - unknown_count / gensim_dict.num_pos
            return cov
```

```
In [ ]: vocab_coverage(dictionary, glove)
Out[ ]: 0.9984438403655991
```

Tokenizer

```
In [ ]: def make_vocab_from_wv(wv, folder=None, special_tokens=None):
            if folder is not None:
                if not os.path.exists(folder):
                    os.mkdir(folder)

            words = wv.index_to_key
            ###########################################################
            if special_tokens is not None:
                to_add = []
                for special_token in special_tokens:
                    if special_token not in words:
                        to_add.append(special_token)
                words = to_add + words

            with open(os.path.join(folder, 'vocab.txt'), 'w') as f:
                for word in words:
                    f.write(f'{word}\n')
```

```
In [ ]: make_vocab_from_wv(glove, 'glove_vocab/', special_tokens=['[PAD]', '[UNK]'])
```

```
In [ ]: glove_tokenizer = BertTokenizer('glove_vocab/vocab.txt')
```

```
In [ ]: glove_tokenizer.encode('alice followed the white rabbit', add_special_tokens=False)
Out[ ]: [7101, 930, 2, 300, 12427]
```

```
In [ ]: len(glove_tokenizer.vocab), len(glove.vectors)
Out[ ]: (400002, 400000)
```

The difference is given by the two special tokens [PAD] and [UNK] and we can add them to our embeddings with all zeros

Special Tokens' Embeddings

```
In [ ]: special_embeddings = np.zeros((2, glove.vector_size))
```

```
In [ ]: extended_embeddings = np.concatenate([special_embeddings, glove.vectors], axis=0)
        extended_embeddings.shape
Out[ ]: (400002, 50)
```

```
In [ ]: alice_idx = glove_tokenizer.encode('alice', add_special_tokens=False)
        np.all(extended_embeddings[alice_idx] == glove['alice'])
Out[ ]: True
```

Model I - GloVe + Classifier

Data Preparation

```
In [ ]: train_sentences = train_dataset['sentence']
        train_labels = train_dataset['labels']
```

```
test_sentences = test_dataset['sentence']
test_labels = test_dataset['labels']
```

```
In [ ]: train_ids = glove_tokenizer(train_sentences,
                                    truncation=True,
                                    padding=True,
                                    max_length=60,
                                    add_special_tokens=False,
                                    return_tensors='pt')['input_ids']
        train_labels = torch.as_tensor(train_labels).float().view(-1, 1)

        test_ids = glove_tokenizer(test_sentences,
                                   truncation=True,
                                   padding=True,
                                   max_length=60,
                                   add_special_tokens=False,
                                   return_tensors='pt')['input_ids']
        test_labels = torch.as_tensor(test_labels).float().view(-1, 1)
```

```
In [ ]: train_tensor_dataset = TensorDataset(train_ids, train_labels)
        generator = torch.Generator()
        train_loader = DataLoader(train_tensor_dataset, batch_size=32, shuffle=True, generator=generator)
        test_tensor_dataset = TensorDataset(test_ids, test_labels)
        test_loader = DataLoader(test_tensor_dataset, batch_size=32)
```

Pre-Trained PyTorch Embeddings

```
In [ ]: extended_embeddings = torch.as_tensor(extended_embeddings).float()
        torch_embeddings = nn.Embedding.from_pretrained(extended_embeddings)
```

```
In [ ]: token_ids, labels = next(iter(train_loader))
        token_ids
```

```
Out[ ]: tensor([[    22,     17,    590,  ...,      0,      0,      0],
                [   934,     36,    104,  ...,      0,      0,      0],
                [   934,    199,    915,  ...,      0,      0,      0],
                ...,
                [   934,     55,    177,  ...,      0,      0,      0],
                [    18,      2, 160237,  ...,      0,      0,      0],
                [    10,      7,    129,  ...,      0,      0,      0]])
```

```
In [ ]: token_embeddings = torch_embeddings(token_ids)
        token_embeddings.shape
```

```
Out[ ]: torch.Size([32, 60, 50])
```

We used the ids to get the embeddings. since we have 32 sentences, of 60 tokens with 50 dimensions each

```
In [ ]: token_embeddings.mean(dim=1)
```

```
Out[ ]: tensor([[ 0.0112, -0.0668, -0.0321,  ..., -0.1178, -0.0145, -0.1179],
                [ 0.0331,  0.0415,  0.0179,  ..., -0.0153,  0.0099,  0.1041],
                [ 0.0342,  0.0460,  0.0229,  ..., -0.0271,  0.0071,  0.0887],
                ...,
                [ 0.0646,  0.0754, -0.0414,  ..., -0.0502,  0.0029,  0.1481],
                [ 0.0038,  0.0089, -0.0008,  ..., -0.0009,  0.0015,  0.0038],
                [ 0.0493,  0.0062, -0.0204,  ..., -0.0507,  0.0320,  0.0624]])
```

For each sentence, we can compute an embedding as an average of the word embeddings and therefore we can use it as features for a classification algorithm

```
In [ ]: '''
        we can use the PyTorch implementation that is nnEmbeddingBag
        '''

        boe_mean = nn.EmbeddingBag.from_pretrained(extended_embeddings, mode='mean')
        boe_mean(token_ids)
```

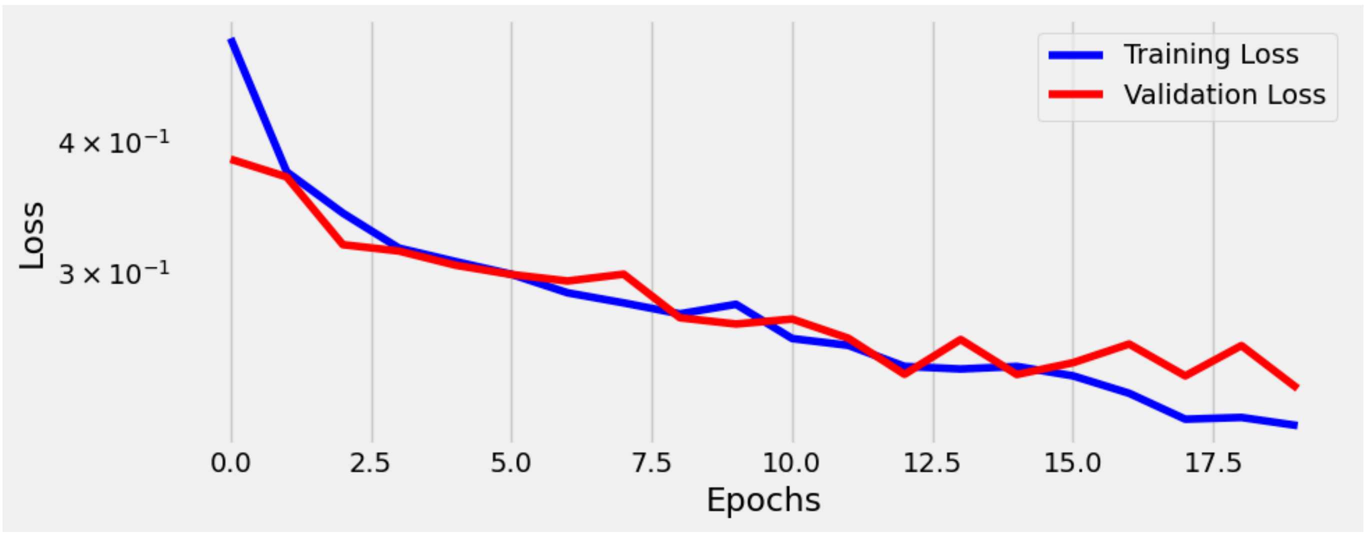
```
Out[ ]: tensor([[ 0.0112, -0.0668, -0.0321,  ..., -0.1178, -0.0145, -0.1179],
                [ 0.0331,  0.0415,  0.0179,  ..., -0.0153,  0.0099,  0.1041],
                [ 0.0342,  0.0460,  0.0229,  ..., -0.0271,  0.0071,  0.0887],
                ...,
                [ 0.0646,  0.0754, -0.0414,  ..., -0.0502,  0.0029,  0.1481],
                [ 0.0038,  0.0089, -0.0008,  ..., -0.0009,  0.0015,  0.0038],
                [ 0.0493,  0.0062, -0.0204,  ..., -0.0507,  0.0320,  0.0624]])
```

Model Configuration & Training

```
In [ ]: extended_embeddings = torch.as_tensor(extended_embeddings).float()
        boe_mean = nn.EmbeddingBag.from_pretrained(
            extended_embeddings, mode='mean'
        )
        torch.manual_seed(41)
        model = nn.Sequential(
            # Embeddings
            boe_mean,
            # Classifier
            nn.Linear(boe_mean.embedding_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )
        loss_fn = nn.BCEWithLogitsLoss()
        optimizer = optim.Adam(model.parameters(), lr=0.01)
```

```
In [ ]: sbs_emb = StepByStep(model, loss_fn, optimizer)
        sbs_emb.set_loaders(train_loader, test_loader)
        sbs_emb.train(20)
```

```
In [ ]: fig = sbs_emb.plot_losses()
```



```
In [ ]: StepByStep.loader_apply(test_loader, sbs_emb.correct)
```

```
Out[ ]: tensor([[396, 444],
                [295, 327]])
```

Model II - GloVe + Transformer

An instance of a transformer encoder, a layer of pre-trained embeddings and the desired number of outputs.

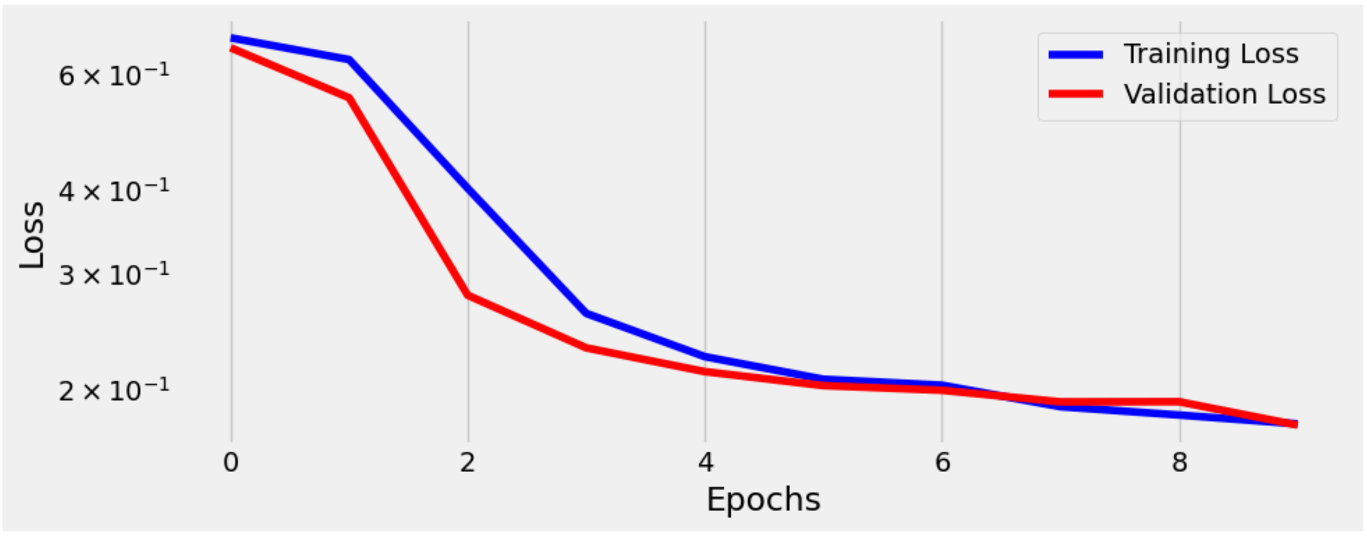
forward takes minibatches of tokenized sentences, preprocess them, encodes them and output the logits.

In [ ]:
```
class TransfClassifier(nn.Module):
    def __init__(self, embedding_layer, encoder, n_outputs):
        super().__init__()
        self.d_model = encoder.d_model
        self.n_outputs = n_outputs
        self.encoder = encoder
        self.mlp = nn.Linear(self.d_model, n_outputs)

        self.embed = embedding_layer
        self.cls_token = nn.Parameter(torch.zeros(1, 1, self.d_model))

    def preprocess(self, X):
        # N, L -> N, L, D
        src = self.embed(X)
        # Special classifier token
        # 1, 1, D -> N, 1, D
        cls_tokens = self.cls_token.expand(X.size(0), -1, -1)
        # Concatenates CLS tokens -> N, 1 + L, D
        src = torch.cat((cls_tokens, src), dim=1)
        return src

    def encode(self, source, source_mask=None):
        # Encoder generates "hidden states"
        states = self.encoder(source, source_mask)
        # Gets state from first token only: [CLS]
        cls_state = states[:, 0]  # N, 1, D
        return cls_state

    @staticmethod
    def source_mask(X):
        cls_mask = torch.ones(X.size(0), 1).type_as(X)
        pad_mask = torch.cat((cls_mask, X > 0), dim=1).bool()
        return pad_mask.unsqueeze(1)

    def forward(self, X):
        src = self.preprocess(X)
        # Featurizer
        cls_state = self.encode(src, self.source_mask(X))
        # Classifier
        out = self.mlp(cls_state) # N, 1, outputs
        return out
```

In [ ]:
```
torch.manual_seed(33)
# Loads the pretrained GloVe embeddings into an embedding layer
torch_embeddings = nn.Embedding.from_pretrained(extended_embeddings)
# Creates a Transformer Encoder
layer = EncoderLayer(n_heads=2, d_model=torch_embeddings.embedding_dim, ff_units=128)
encoder = EncoderTransf(layer, n_layers=1)
# Uses both layers above to build our model
model = TransfClassifier(torch_embeddings, encoder, n_outputs=1)
loss_fn = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)
```

In [ ]:
```
sbs_transf = StepByStep(model, loss_fn, optimizer)
sbs_transf.set_loaders(train_loader, test_loader)
sbs_transf.train(10)
```

In [ ]:
```
fig = sbs_transf.plot_losses()
```



```
In [ ]: StepByStep.loader_apply(test_loader, sbs_transf.correct)

Out[ ]: tensor([[417, 444],
                [291, 327]])
```

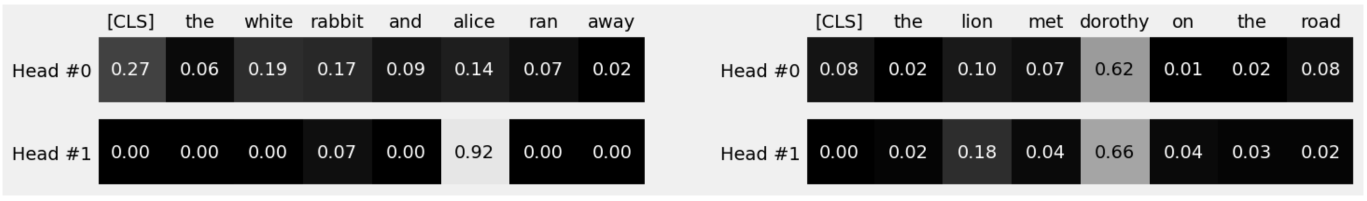
Visualizing Attention

```
In [ ]: sentences = ['The white rabbit and Alice ran away', 'The lion met Dorothy on the road']
        inputs = glove_tokenizer(sentences, add_special_tokens=False, return_tensors='pt')['input_ids']
        inputs = inputs.to(sbs_transf.device)
        inputs

Out[ ]: tensor([[    2,   300, 12427,     7,  7101,  1423,   422],
                [    2,  6659,   811, 11238,    15,     2,   588]], device='cuda:0')
```

```
In [ ]: sbs_transf.model.eval()
        out = sbs_transf.model(inputs)
        # our model outputs logits, so we turn them into probs
        torch.sigmoid(out)

Out[ ]: tensor([[0.9885],
                [0.0100]], device='cuda:0', grad_fn=<SigmoidBackward0>)
```

```
In [ ]: alphas = sbs_transf.model.encoder.layers[0].self_attn_heads.alphas
        alphas[:, :, 0, :].squeeze()

Out[ ]: tensor([[[2.7092e-01, 5.7875e-02, 1.8512e-01, 1.7325e-01, 8.6493e-02,
                  1.3617e-01, 7.4261e-02, 1.5902e-02],
                 [3.8458e-05, 2.8619e-03, 4.5050e-03, 7.1031e-02, 2.7900e-03,
                  9.1666e-01, 9.2231e-04, 1.1937e-03]],

                [[7.9923e-02, 1.7073e-02, 1.0265e-01, 7.3170e-02, 6.2225e-01,
                  1.1677e-02, 1.5310e-02, 7.7946e-02],
                 [2.7508e-04, 2.0471e-02, 1.8471e-01, 4.1392e-02, 6.6281e-01,
                  4.0948e-02, 2.5272e-02, 2.4122e-02]]], device='cuda:0')
```

```
In [ ]: tokens = [['[CLS]'] + glove_tokenizer.tokenize(sent) for sent in sentences]
        fig = plot_attention(tokens, alphas)

        /content/plots/chapter11.py:50: UserWarning: set_ticklabels() should only be used with a fixed number of tick
        s, i.e. after set_ticks() or using a FixedLocator.
          ax.set_yticklabels(['', f'Head #{i}', ''])
        /content/plots/chapter11.py:50: UserWarning: set_ticklabels() should only be used with a fixed number of tick
        s, i.e. after set_ticks() or using a FixedLocator.
          ax.set_yticklabels(['', f'Head #{i}', ''])
        /content/plots/chapter11.py:50: UserWarning: set_ticklabels() should only be used with a fixed number of tick
        s, i.e. after set_ticks() or using a FixedLocator.
          ax.set_yticklabels(['', f'Head #{i}', ''])
        /content/plots/chapter11.py:50: UserWarning: set_ticklabels() should only be used with a fixed number of tick
        s, i.e. after set_ticks() or using a FixedLocator.
          ax.set_yticklabels(['', f'Head #{i}', ''])
```



Contextual Word Embeddings

ELMo

Sentence 1

The Hatter was the first to break the silence. `What day of the month is it?' he said, turning to Alice: he had taken his **watch** out of his pocket, and was looking at it uneasily, shaking it every now and then, and holding it to his ear.

Sentence 2

Alice thought this a very curious thing, and she went nearer to **watch** them, and just as she came up to them she heard one of them say, `Look out now, Five! Don't go splashing paint over me like that!

Watch in these two sentences has a different meaning noun and verb. Probabily, the single word embedding is not enough... we need to consider the context, the sentence itself to represent the word.

These are called contextual word embeddings where we don't have a look-up table between every combination of word and context, but the embeddings are the outputs of a model :-)

ELMo takes into account also the context.

It is a two layer bidirectional LSTM encoder using 4096 dimensions for its cell states

The representations are char-based, so it can easily handle unkown words

Flair is a NLP (yet another) library built on top of pytorch that offers word embeddings and document embeddings for ELMo and BERT as well as GloVe.

In [ ]:
```
watch1 = """
The Hatter was the first to break the silence. `What day of the month is it?' he said, turning to Alice:  he h
ad taken his watch out of his pocket, and was looking at it uneasily, shaking it every now and then, and holdi
ng it to his ear.
"""

watch2 = """
Alice thought this a very curious thing, and she went nearer to watch them, and just as she came up to them sh
e heard one of them say, `Look out now, Five!  Don't go splashing paint over me like that!
"""

sentences = [watch1, watch2]
```

In [ ]:
```
from flair.data import Sentence

flair_sentences = [Sentence(s) for s in sentences]
flair_sentences[0]
```

Out[ ]:
```
Sentence[58]: " The Hatter was the first to break the silence. `What day of the month is it?' he said, turning
to Alice:  he had taken his watch out of his pocket, and was looking at it uneasily, shaking it every now and
then, and holding it to his ear."
```

In [ ]:
```
flair_sentences[0].tokens[31]
```

Out[ ]:
```
Token[31]: "watch"
```

In [ ]:
```
flair_sentences[0].tokens[0]
```

Out[ ]:
```
Token[0]: "The"
```

In [ ]:
```
flair_sentences[0].get_token(1)
```

Out[ ]:
```
Token[0]: "The"
```

In [ ]:
```
from flair.embeddings import FlairEmbeddings
flair_emb = FlairEmbeddings('news-forward')
```

In [ ]:
```
flair_emb.embed(flair_sentences)
```

Out[ ]:
```
[Sentence[58]: " The Hatter was the first to break the silence. `What day of the month is it?' he said, turnin
g to Alice:  he had taken his watch out of his pocket, and was looking at it uneasily, shaking it every now an
d then, and holding it to his ear.",
 Sentence[48]: " Alice thought this a very curious thing, and she went nearer to watch them, and just as she c
ame up to them she heard one of them say, `Look out now, Five!  Don't go splashing paint over me like that!"]
```

In [ ]:
```
token_watch1 = flair_sentences[0].tokens[31]
token_watch2 = flair_sentences[1].tokens[13]
token_watch1, token_watch2
```
Out[ ]:
```
(Token[31]: "watch", Token[13]: "watch")
```
In [ ]:
```
token_watch1.embedding, token_watch2.embedding
```
Out[ ]:
```
(tensor([-0.0007, -0.0057,  0.0187,  ..., -0.0051, -0.0022,  0.0026],
        device='cuda:0'),
 tensor([-0.0011, -0.0037,  0.1031,  ..., -0.0079, -0.0044,  0.0002],
        device='cuda:0'))
```
In [ ]:
```
similarity = nn.CosineSimilarity(dim=0, eps=1e-6)
similarity(token_watch1.embedding, token_watch2.embedding)
```
Out[ ]:
```
tensor(0.5003, device='cuda:0')
```
In [ ]:
```
def get_embeddings(embeddings, sentence):
    sent = Sentence(sentence)
    embeddings.embed(sent)
    return torch.stack([token.embedding for token in sent.tokens]).float()
```
In [ ]:
```
get_embeddings(flair_emb, watch1)
```
Out[ ]:
```
tensor([[-2.0933e-03,  5.0415e-04,  4.6944e-02,  ..., -4.4014e-04,
         -3.9301e-02,  1.0601e-02],
        [ 3.2224e-04,  1.4939e-03,  2.5947e-02,  ..., -9.2415e-04,
         -1.4211e-02,  3.0942e-03],
        [ 1.7407e-03, -1.8324e-04,  5.3375e-02,  ...,  4.5165e-04,
          5.4484e-02,  9.7125e-03],
        ...,
        [ 2.2250e-03,  2.2138e-04,  4.4385e-02,  ..., -3.9766e-03,
         -7.9449e-04,  2.2444e-02],
        [-2.2455e-03,  9.7197e-04,  3.4624e-02,  ..., -7.1559e-05,
          1.2647e-02,  1.3210e-02],
        [ 1.0997e-03,  8.4726e-05,  1.5534e-01,  ..., -1.5737e-04,
          9.5801e-04,  5.3785e-03]], device='cuda:0')
```

GloVe

Let't see the differences between GloVe and the contexutal embedding.

What do you expect?

In [ ]:
```
from flair.embeddings import WordEmbeddings
glove_embedding = WordEmbeddings('glove')
```
In [ ]:
```
new_flair_sentences = [Sentence(s) for s in sentences]
glove_embedding.embed(new_flair_sentences)
```
Out[ ]:
```
[Sentence[58]: " The Hatter was the first to break the silence. `What day of the month is it?' he said, turnin
g to Alice:  he had taken his watch out of his pocket, and was looking at it uneasily, shaking it every now an
d then, and holding it to his ear.",
 Sentence[48]: " Alice thought this a very curious thing, and she went nearer to watch them, and just as she c
ame up to them she heard one of them say, `Look out now, Five!  Don't go splashing paint over me like that!"]
```
In [ ]:
```
torch.all(new_flair_sentences[0].tokens[31].embedding == new_flair_sentences[1].tokens[13].embedding)
```
Out[ ]:
```
tensor(True, device='cuda:0')
```

BERT

In [ ]:
```
from flair.embeddings import TransformerWordEmbeddings
bert_flair = TransformerWordEmbeddings('bert-base-uncased', layers='-1')
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/file_download.py:942: FutureWarning: `resume_download`
is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you want to forc
e a new download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/setting
s/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

In [ ]:
```
embed1 = get_embeddings(bert_flair, watch1)
embed2 = get_embeddings(bert_flair, watch2)
embed2
```

Out[ ]:
```
tensor([[ 0.6554, -0.3799, -0.2842,  ...,  0.0601,  0.8865,  0.4760],
        [-0.1459, -0.0204, -0.0615,  ..., -0.0257,  0.5052,  0.3324],
        [-0.0436, -0.0400, -0.0135,  ...,  0.3920,  0.5231,  0.9067],
        ...,
        [-0.2582,  0.6933,  0.2688,  ..., -0.0325,  0.0772,  0.2187],
        [-0.1868,  0.6398, -0.8127,  ..., -0.5957,  0.2793,  0.1880],
        [-0.1021,  0.5222, -0.7142,  ...,  0.3644,  0.0600, -0.1419]],
       device='cuda:0')
```

In [ ]:
```
bert_watch1 = embed1[31]
bert_watch2 = embed2[13]
bert_watch1, bert_watch2
```

```
Out[ ]: (tensor([ 8.5760e-01,  3.5888e-01, -3.7825e-01, -8.3564e-01,  1.3528e+00,
                  1.0204e-01, -8.2618e-01,  3.1570e-01,  3.1091e-01, -4.2653e-01,
                  1.6632e-01,  7.2440e-02, -1.0276e+00,  4.4680e-01, -1.4549e-01,
                  1.8315e-01,  3.7484e-01, -4.0245e-01, -1.4190e-01, -5.1596e-01,
                  1.1984e+00,  6.8202e-01,  1.1028e+00, -4.3956e-02,  1.6932e-02,
                  7.4420e-01, -1.0604e-01,  3.8719e-01,  7.3662e-02,  2.2424e-01,
                 -5.0282e-02,  1.9586e-01,  1.0773e+00,  1.0341e+00,  4.4217e-01,
                 -6.5567e-01, -5.6308e-01, -1.1827e-01,  1.2862e-01,  4.6581e-01,
                 -7.1305e-01, -1.5682e-01, -1.3552e-01,  4.5852e-02,  8.2852e-03,
                  5.2762e-01,  1.6906e+00,  5.5388e-01, -7.3777e-02,  5.1504e-01,
                 -2.0619e-01, -2.9503e-03,  2.5389e-01, -1.5379e-01,  8.4900e-01,
                  6.5440e-01, -3.2925e-01, -1.1198e+00,  2.4251e-01,  1.5586e-01,
                  5.7514e-01,  8.5832e-02,  3.3306e-01, -3.9344e-01, -4.1766e-01,
                  1.0790e+00,  1.1719e-02,  7.0214e-01, -6.0665e-01,  7.3702e-01,
                  6.8223e-02, -3.7110e-01, -7.7589e-01, -1.3057e-01,  1.7179e-01,
                  6.2051e-01, -6.4163e-01,  4.2881e-01,  5.7445e-01,  3.9801e-02,
                 -2.1279e-01, -5.1984e-01,  7.5919e-01,  3.0600e-01, -3.3987e-01,
                 -3.0427e-01, -5.7486e-01, -5.2273e-02, -6.8957e-01,  7.3935e-01,
                 -1.6545e-01, -8.2786e-01,  3.4169e-02,  7.1547e-01, -5.6962e-01,
                 -7.8686e-01,  3.6913e-01, -1.9961e-02, -5.7287e-01, -2.6407e-01,
                  8.5581e-01, -6.1419e-02,  3.8184e-01, -9.1566e-03, -1.0273e+00,
                 -5.3303e-01,  4.4014e-01,  1.7875e-01,  9.1851e-01,  5.9216e-01,
                 -4.8853e-01, -3.9106e-01, -8.8284e-02, -7.1935e-01, -3.8647e-01,
                  9.9550e-02,  2.2983e-01, -1.5325e+00,  6.5409e-02, -1.0896e+00,
                  1.2108e-01, -1.9921e-01, -2.5195e-01,  1.4421e-01,  2.7531e-02,
                 -3.0479e-01, -3.8533e-01, -1.1540e-01, -8.3979e-01, -7.7467e-01,
                  6.5181e-01,  3.1980e-01,  4.7616e-01, -8.2181e-01, -4.4323e-01,
                  1.2445e-01, -2.9082e-01, -4.2669e-01, -6.8031e-01,  1.1736e+00,
                  1.0199e+00,  7.3959e-01, -6.1510e-01,  1.0691e+00,  5.6806e-01,
                  9.4541e-01, -9.8110e-01, -7.1101e-01,  3.6151e-01,  2.0443e-01,
                 -5.0706e-02, -2.8712e-01,  1.2332e-01,  3.9614e-01, -4.4268e-01,
                 -5.5477e-01,  2.8325e-01,  5.5901e-01, -3.8944e-01, -4.2003e-01,
                 -1.2470e-01, -9.5830e-01, -3.4775e-02,  1.0381e+00, -2.2607e-02,
                  4.7267e-01,  8.1840e-01,  9.0778e-01, -1.9271e-02,  9.3007e-03,
                 -2.9097e-01,  5.1579e-01,  6.5791e-01,  3.1353e-01,  4.7983e-01,
                  6.2282e-01,  1.4813e-01, -3.8600e-01,  1.8680e-01, -7.3365e-01,
                 -5.2868e-01, -3.5756e-01,  1.4778e+00,  1.7065e+00,  5.2843e-01,
                  4.3788e-01, -6.0783e-01, -2.0891e-01, -2.6378e-01, -3.5563e-01,
                 -1.9136e-01,  4.4895e-01, -2.1979e-01, -2.2114e-01, -6.3080e-01,
                 -1.1537e+00,  3.8770e-01, -6.5096e-01, -1.4381e-01, -3.3387e-01,
                 -4.8165e-01, -2.2851e-01,  4.4550e-01, -1.4106e-01, -1.9820e-01,
                  1.2682e-01,  5.7340e-01,  5.2960e-01,  3.8535e-01, -1.8421e-01,
                 -2.7734e-01,  2.1106e-02, -5.1269e-01,  1.2724e-01, -5.2737e-01,
                  1.9794e-01,  6.6517e-01, -4.9908e-01, -5.2315e-01, -3.3081e-01,
                 -6.9502e-01, -2.1738e-01, -9.1546e-01,  8.4446e-01, -8.6271e-01,
                  9.8701e-01, -5.3786e-01,  2.2177e-01, -7.9477e-01,  1.6505e-01,
                 -3.9847e-01, -2.9883e-01,  2.0877e-01,  2.4684e-01,  1.0068e-01,
                  1.6014e-01, -1.1922e+00, -1.5679e-01, -2.0235e-03, -5.7467e-01,
                  4.6138e-03, -2.6884e-01,  4.2084e-01, -7.2791e-03,  3.4696e-01,
                  5.3609e-01,  2.1878e-01,  3.3520e-01, -2.8570e-01, -5.0958e-01,
                 -3.9454e-01, -8.9165e-01,  1.2181e+00, -6.8443e-01,  1.9280e-01,
                 -2.1449e-01,  4.6196e-02,  5.2594e-01, -4.3565e-02,  2.4555e-01,
                  1.9387e+00,  1.5278e-01,  9.9872e-01,  4.8577e-01, -4.1082e-01,
                 -8.3016e-01,  2.7045e-01,  4.8655e-01, -2.4722e-01, -1.8512e-01,
                  2.0524e-01, -6.6171e-02,  2.3728e-01,  1.2404e+00, -2.6077e-01,
                 -1.6077e-01,  1.3507e-01,  1.1847e-01, -2.9235e-01,  5.5437e-01,
                 -1.8560e-01,  1.3344e+00, -3.6667e-01, -3.7478e-01, -5.5665e-02,
                 -3.4930e-01,  8.5271e-01,  1.5741e+00,  6.0562e-01,  4.3780e-01,
                 -1.0298e+00,  4.0719e-01, -8.2617e-02,  2.7545e-02, -4.9199e-01,
                  7.3797e-01,  2.4101e-01,  9.6176e-01,  6.4367e-01,  1.1653e-01,
                  9.4047e-01, -2.6082e-01,  1.0976e+00, -1.1490e-01, -9.5253e-01,
                  5.8644e-02, -2.3060e-01, -6.7886e-02, -3.0703e+00,  2.4538e-02,
                  2.7582e-01, -4.3797e-01, -7.8420e-02,  4.2382e-01,  1.0763e+00,
                  1.8140e-01, -1.0572e+00, -3.9120e-01, -9.3492e-02, -3.7970e-01,
                  5.3947e-02,  1.1576e+00,  8.4130e-01,  1.8007e-01, -5.2631e-01,
                 -2.0291e-01, -1.9092e-02,  6.1720e-01,  1.1918e-01, -1.4457e+00,
                 -2.5592e-01, -1.2389e-01,  2.3928e-01,  1.1959e+00, -5.4284e-01,
                 -8.6952e-01, -4.8750e-01,  2.3500e-01,  1.3830e-02, -2.6659e-01,
                  9.7897e-01, -8.5489e-01,  3.1946e-01, -4.9998e-01, -3.2241e-01,
                 -1.3864e-01,  1.2900e-01, -7.5765e-01, -4.5315e-01,  2.9473e-01,
                  8.7803e-02, -1.3806e-01,  4.5430e-01,  8.0650e-02, -4.1994e-01,
                  5.8679e-03, -7.0910e-02,  3.8368e-01, -1.3059e-01,  8.7690e-01,
                 -6.1343e-01, -6.1015e-01, -7.0236e-01,  3.7512e-01,  3.3982e-01,
                  8.0856e-01, -1.6385e-01, -1.0569e-01,  6.1967e-01, -5.8364e-02,
                  2.3294e-01, -2.7713e-01,  5.2177e-01, -1.7158e-01, -1.0602e+00,
                 -4.3479e-01, -6.6953e-02,  3.6794e-01, -2.3135e-01, -2.0400e-01,
                 -5.3282e-01, -1.2286e+00, -2.4328e-01, -2.8658e-01,  1.3142e+00,
                  3.1902e-01, -9.1673e-02, -6.9225e-01, -3.0026e-01, -9.0491e-01,
                  8.4392e-02,  2.1298e-01,  8.2160e-01, -6.7558e-01, -8.9888e-01,
                  3.5113e-01, -5.2840e-01, -3.1060e-02, -2.8948e-01,  5.1492e-01,
                 -2.2329e-02,  1.6455e-01, -1.3406e-01,  7.9522e-01, -1.0633e-01,
                  4.0473e-01,  3.6492e-01, -6.1566e-01, -5.1250e-01, -5.4742e-01,
          3.2135e-02, -4.5447e-02,  5.3155e-02,  7.0305e-01, -1.5488e+00,
          6.2800e-01, -4.1181e-01,  3.1977e-01,  3.9906e-01, -1.7447e-01,
          1.0357e+00, -5.4064e-01,  2.1086e-01, -5.4963e-01, -3.4768e-02,
         -1.6814e-01,  4.8931e-01,  1.6012e-01, -6.6077e-01, -6.7637e-02,
          9.2604e-02, -1.1114e+00,  4.6336e-01, -3.5745e-01, -3.2055e-02,
          3.4265e-01,  3.5538e-01,  6.7498e-01, -2.0282e-01,  2.1987e-02,
         -1.0548e+00,  9.8446e-01, -5.6733e-01,  6.9380e-02, -4.7736e-01,
         -9.9546e-01, -3.5254e-02,  4.6173e-01,  7.1046e-01,  3.3707e-01,
         -8.1136e-02, -4.8355e-01,  5.4515e-01,  3.7669e-02,  5.2528e-01,
          4.1853e-02, -1.2564e+00, -3.2226e-01, -8.2649e-01, -7.5457e-02,
         -7.5999e-01, -1.8107e-01,  1.1781e+00,  7.7896e-01,  6.6844e-01,
         -7.6050e-01, -4.5029e-01, -5.4186e-01,  2.7288e-03,  2.3229e-01,
         -6.9320e-01, -4.4791e-02, -7.4108e-02,  9.3096e-01, -4.1993e-01,
         -7.3685e-01,  6.3003e-02,  4.0879e-01, -7.2932e-01, -1.6402e-01,
         -2.7400e-01,  7.5049e-01, -6.5255e-01, -4.3226e-01, -6.2632e-02,
         -8.2434e-01, -9.1996e-01,  5.1996e-01,  3.3342e-01,  3.0549e-03,
         -3.7381e-01,  8.1910e-01, -2.5230e-01, -8.2635e-01,  3.8007e-01,
         -1.0642e+00, -7.9937e-01,  1.0933e-01, -2.3652e-01,  4.5540e-01,
         -2.9788e-01, -1.5564e+00, -2.3166e-01, -1.1862e+00, -2.9165e-01,
         -2.1934e-01,  1.1169e-01,  1.3712e-01, -5.1983e-01, -2.5509e-01,
         -4.4754e-01, -4.7948e-01,  4.6833e-01, -1.8625e-01,  6.9716e-01,
         -1.1866e-01,  6.1084e-01,  6.3114e-01, -8.4579e-01, -4.6913e-01,
          9.2668e-02,  2.9336e-02,  2.5508e-01,  4.5543e-01,  1.8251e-01,
         -1.1855e-01, -1.2805e-01,  8.3519e-02,  1.7325e-01,  6.1112e-01,
         -3.9921e-01,  1.6798e-01, -8.4275e-02,  3.9789e-01,  9.2255e-01,
          7.7275e-02, -4.0163e-01,  3.1146e-01, -3.0563e-01, -5.8679e-01,
          4.1130e-01, -4.3300e-01, -1.2496e+00, -8.5653e-01, -2.3101e-01,
         -1.2980e+00, -2.7850e-01,  1.3634e-01,  3.4896e-01,  4.5002e-01,
         -6.9828e-01,  6.2787e-01, -2.2598e-01, -1.7731e-01, -3.2118e-02,
          4.8136e-01,  5.0901e-01,  5.0199e-01,  3.6993e-01, -4.5981e-01,
         -6.7486e-02, -8.6722e-01, -5.7723e-01, -3.2765e-01,  7.8836e-01,
          4.0974e-01, -1.8077e-01, -8.4644e-01, -2.5588e-01,  1.0655e+00,
         -3.7329e-01, -4.9543e-01, -2.4806e-01, -3.7839e-02,  9.1718e-02,
         -1.5725e-01,  2.4216e-01,  6.5864e-02,  1.3641e+00, -1.2707e+00,
          7.8174e-02, -1.0401e+00, -3.9388e-01, -2.5627e-01, -4.9158e-01,
         -4.0819e-01,  7.3981e-01, -4.4651e-01, -3.1208e-01,  6.2193e-01,
          1.1230e+00, -4.4226e-02, -1.2044e+00,  2.2500e-01, -9.1115e-02,
          3.5365e-01, -3.7870e-01, -1.0870e-01,  3.2082e-01,  1.2314e-01,
         -2.1645e-02,  1.1413e+00,  1.3573e-02, -9.9737e-02,  9.3759e-01,
          5.0417e-02, -4.9671e-01,  9.0127e-01,  1.4178e-01,  2.0027e-01,
          6.3084e-01, -2.0751e-02,  3.0619e-01,  4.5247e-01, -7.6788e-01,
         -7.2624e-01, -5.2419e-01,  6.0564e-01, -1.9887e-02, -2.9045e-01,
          4.0384e-01,  9.0467e-01, -9.9711e-01,  1.0205e-02,  1.7958e-01,
          3.5786e-02, -2.3696e-01,  6.4812e-01,  1.5352e-01,  3.1153e-02,
          1.3805e-01, -1.4471e-01, -9.9713e-01,  1.7288e-01,  1.8074e-01,
         -2.8294e-02,  7.8877e-01,  3.7700e-01, -7.0563e-02,  1.0712e+00,
         -4.2392e-01,  4.1656e-01, -6.8028e-01,  1.6911e-01,  2.6617e-01,
         -1.1312e+00,  1.6047e-01, -4.5070e-01,  4.9935e-01,  3.4413e-01,
          6.4540e-01, -2.6726e-01,  1.5033e-01,  4.2149e-01,  3.4961e-01,
         -1.5505e-01,  1.5471e+00,  7.7871e-01, -2.7912e-01, -4.5703e-01,
         -7.9149e-01, -5.7845e-01, -2.4879e-01, -6.0028e-01,  6.9574e-02,
         -1.9024e-02,  2.0337e-01,  1.3373e+00,  1.7257e-01, -3.3319e-01,
          1.4024e-01, -2.2573e-02, -2.2282e-01,  3.4986e-02, -1.4985e-01,
          5.2162e-02,  1.5562e-01, -2.3856e-01, -7.4214e-02, -2.1547e-01,
          7.6900e-01,  1.1663e-01,  5.3913e-02, -2.6870e-01, -2.3951e-01,
          8.8452e-01,  3.2750e-01, -7.8228e-02,  4.5901e-01, -4.3255e-01,
         -4.4316e-01, -3.2504e-01, -1.4550e+00, -8.6594e-02, -7.3577e-02,
         -2.0595e-01, -6.1714e-01, -7.1308e-01,  8.2567e-01,  1.6622e-01,
         -8.5806e-01, -6.5291e-01, -1.5518e-02, -6.2837e-01,  5.6276e-01,
          9.6669e-02,  4.9524e-01,  3.4259e-01, -6.7065e-01, -4.1591e-01,
         -3.3399e-01, -7.4808e-01,  3.2285e-01, -5.1326e-02,  4.1144e-01,
          2.6481e-01, -9.0313e-01, -4.4464e-01,  6.6254e-02,  4.1635e-01,
         -7.5472e-01,  1.2165e+00, -4.2961e-01,  6.3139e-01, -2.1192e-02,
         -4.6616e-01, -1.4575e-01, -7.4167e-01, -2.7592e-01, -4.6199e-02,
          4.5393e-01,  4.7207e-01, -1.7622e-01, -6.7738e-01,  3.7787e-01,
          5.3374e-01,  3.6832e-01,  1.3724e-01, -1.1172e+00, -2.5947e-01,
          1.1888e+00, -1.0117e-01, -4.8335e-01,  5.8897e-02,  1.6610e-01,
          6.6885e-01, -9.0577e-01, -2.3182e-01,  5.9258e-01,  1.1081e-01,
          1.8942e-01,  6.5041e-01,  4.6973e-01, -3.1615e-01, -5.9470e-01,
         -5.1061e-01, -9.7604e-01,  6.0229e-01, -6.3250e-01, -4.2595e-01,
         -5.7413e-01, -7.1769e-01,  9.2507e-01, -6.2844e-01,  1.8694e-01,
          2.0768e-01,  1.1880e-01,  4.1445e-01], device='cuda:0'),
 tensor([-9.8449e-02,  1.4698e+00,  2.8573e-01, -3.9569e-01,  9.6890e-02,
          8.6436e-01, -3.7401e-01,  2.1752e-01, -2.5937e-01, -4.2487e-01,
         -6.1410e-02,  3.3272e-01, -1.9102e-01, -2.9803e-02, -1.7782e-01,
          7.1223e-01, -3.6092e-01,  4.7531e-01,  1.8901e-01, -1.9464e-01,
          5.5041e-01, -2.0075e-01, -4.9742e-01, -1.3700e-02,  6.5577e-02,
          2.2761e-01,  7.6017e-01, -7.9264e-02,  2.7941e-01, -5.4700e-02,
         -2.6595e-01,  5.5095e-01, -2.1662e-01,  1.4398e-01, -3.8731e-01,
         -1.9042e-01,  1.3155e-01,  2.2447e-01,  2.9488e-01,  2.7523e-01,
         -2.7458e-01, -5.0480e-01, -3.9631e-01,  3.0837e-01, -6.3027e-01,
         -3.5416e-01,  1.1856e+00,  8.9831e-02,  4.4822e-01,  4.0406e-01,
-3.4759e-01,  5.6113e-01,  1.0385e+00,  1.5012e-01,  3.7633e-01,
-3.9084e-01,  1.0627e-01, -5.0344e-02,  2.2779e-01, -5.2758e-01,
-1.9242e-01,  1.8374e-02,  5.2802e-01,  1.6686e-01, -7.3694e-01,
 4.7303e-01, -4.7016e-01,  1.6289e-01,  9.7774e-02,  1.2824e-01,
-2.8870e-01, -3.9116e-01, -1.8324e-01,  5.2460e-01, -1.9830e-01,
 5.4971e-01,  4.1160e-02, -3.3559e-01,  2.1168e-02, -8.5710e-02,
 1.3109e-01, -3.6348e-01,  2.7382e-01,  1.1263e+00, -2.5038e-01,
-6.4161e-01, -3.2357e-01, -1.3938e-01, -1.0110e+00,  7.3331e-01,
 1.0102e-01,  1.5387e-01, -2.9724e-01,  2.5424e-01,  3.2296e-01,
-3.5495e-01,  1.2272e-01,  9.6730e-02,  2.1486e-02,  9.1426e-02,
-1.4177e-01,  5.2628e-01,  2.1227e-01, -1.7098e-01, -3.8352e-01,
 1.6601e-01,  4.4298e-01,  6.3218e-03, -5.9510e-01,  3.5965e-01,
-7.1792e-01, -3.6388e-01, -7.0435e-02,  1.8896e-01,  5.4331e-03,
 1.1766e-02,  1.0752e-01, -5.5095e-01, -6.7836e-02,  3.5240e-01,
-2.6229e-01,  3.7230e-01, -8.1172e-02,  4.8515e-01,  1.0485e-02,
-2.9454e-01, -3.1111e-01, -2.6310e-01,  6.3381e-02, -1.0854e-01,
 1.3688e+00,  2.9465e-01,  2.0623e-01,  3.8317e-02,  5.4437e-02,
-1.5726e+00,  3.7239e-01, -6.0964e-01, -8.8024e-01,  2.4682e-01,
 6.7070e-01, -5.6562e-01,  2.1737e-01,  8.4505e-02, -2.6882e-01,
 1.4112e-01, -8.0701e-01, -1.1845e-01, -3.2562e-01,  7.8287e-01,
-1.8123e-01, -7.1260e-01,  2.0073e-01, -2.2902e-01, -1.7190e-01,
 8.2328e-02,  3.2728e-01,  3.3844e-01, -4.3359e-01, -2.0109e-01,
 5.2177e-01, -2.1286e-01, -5.4271e-01,  4.2771e-01, -2.8439e-02,
 6.8534e-01,  3.0166e-01,  3.3880e-01, -7.2538e-01,  4.7892e-02,
-5.7754e-01,  4.3418e-01,  6.2832e-01,  1.1230e-01,  1.6312e-02,
-3.1558e-01,  3.0021e-01, -3.4793e-01, -4.2287e-01,  8.9215e-02,
-1.7020e-01,  5.5911e-01,  4.3507e-01,  1.0352e+00, -6.0532e-02,
 6.4033e-02,  6.3265e-01,  5.8860e-01, -6.2739e-01, -3.6926e-01,
-1.3001e-01,  3.0337e-01, -7.3679e-01, -1.0968e+00, -4.9620e-01,
-2.4885e-02,  6.0554e-02, -2.8982e-01, -5.5470e-01,  9.2083e-02,
-3.5928e-01, -6.8506e-01,  7.0259e-01,  7.4057e-01,  4.1953e-02,
-2.2261e-01,  3.1581e-02, -5.7034e-01, -1.5258e-01, -1.9746e-01,
-5.7486e-01, -6.6855e-01, -5.3475e-01,  2.1482e-01, -8.8653e-01,
 9.2318e-02, -2.2077e-02,  9.8558e-01, -8.5093e-01, -1.6057e-02,
-1.1169e-01, -2.8778e-01, -4.2301e-01,  7.7859e-02,  6.9498e-02,
 4.2390e-01, -7.0422e-02,  7.7603e-02, -4.0021e-01, -2.4131e-01,
 1.3328e-01, -1.1587e-01,  9.0171e-01, -3.4312e-01,  6.0788e-01,
 3.1653e-01, -2.4758e-01,  4.5196e-02,  4.9370e-01,  3.8995e-01,
-3.4692e-01,  3.3056e-02,  2.0106e-01,  4.0234e-01,  1.2993e-01,
 2.9119e-01,  9.2159e-01, -4.1758e-01,  1.5004e-02,  5.2865e-02,
 6.0922e-01,  7.6300e-02,  2.6940e-01, -6.3987e-01,  1.7187e-01,
-1.4997e-01, -2.2630e-01, -5.7578e-01, -5.4688e-02, -1.4096e-01,
-1.8318e-01,  1.6508e-01,  5.1326e-01,  4.4380e-01, -2.8080e-01,
-4.5417e-01, -3.0488e-01,  4.8724e-01, -1.7721e-01, -2.2289e-01,
 6.0853e-01, -2.2062e-04,  5.6217e-01,  1.1224e+00, -1.4180e+00,
-3.6117e-01,  1.5190e-01, -1.7284e-01, -1.7632e-02, -2.1651e-01,
-8.6195e-01,  4.2032e-02, -4.6597e-01,  7.2990e-02, -6.2120e-02,
-3.9781e-01,  3.5995e-01, -3.4646e-01, -3.0336e-01, -1.5005e-02,
 3.3628e-01,  5.7878e-02, -3.3315e-01,  1.1097e-01,  3.0777e-02,
-7.1115e-01,  3.6039e-01,  1.6007e-01,  3.4098e-01, -5.3309e-01,
 5.6319e-01,  2.1157e-01,  2.6252e-01,  4.2321e-01, -5.2220e-01,
 2.4814e-01, -3.6991e-01, -5.4816e-01, -5.3290e+00,  1.0130e-01,
-2.4280e-01,  2.6960e-01,  6.2188e-01,  3.4420e-01,  2.2724e-01,
 1.2104e-01, -3.0778e-01, -1.3375e-01,  1.2070e-01,  6.9458e-01,
-1.7730e-01,  6.3572e-01,  3.9321e-01,  1.0829e-03, -4.1992e-01,
-3.1437e-01,  4.6219e-01,  3.0782e-01, -2.4236e-01, -6.3911e-01,
-7.2800e-03, -2.1637e-02,  2.9671e-01,  8.6041e-01, -1.6743e-01,
-5.2726e-01, -7.3631e-01, -4.9699e-01, -4.0285e-02, -5.5160e-02,
-2.8419e-01,  3.4066e-01,  1.9508e-01, -3.1847e-01,  1.1065e-01,
-1.3114e+00, -6.5144e-02, -5.2608e-01, -2.2926e-01, -4.9479e-01,
-3.6531e-01, -1.5898e-01,  9.3911e-02,  1.2114e-01, -1.5592e-01,
 1.9144e-01, -7.1318e-01, -1.7203e-01, -6.8007e-02,  4.2680e-01,
 2.7249e-01, -8.1866e-02, -7.7034e-01,  4.4868e-02,  6.1248e-01,
 6.4211e-01, -4.2664e-01, -4.1450e-01, -4.6396e-02, -5.7337e-01,
 4.2681e-01, -4.0819e-02,  2.8339e-02, -3.5921e-01, -2.4894e-01,
-4.9875e-02,  5.8249e-01, -1.4033e-02, -3.2829e-01,  1.0102e-01,
-4.5791e-01, -1.0023e+00, -2.6306e-01, -2.5233e-01,  6.0858e-01,
 2.0588e-01, -2.5912e-01, -1.0158e-01, -3.4643e-01, -1.6082e-01,
-3.1902e-01,  5.3895e-01,  2.7880e-01, -6.8466e-01, -7.6329e-02,
 9.4314e-02,  1.3531e-01, -8.8651e-01, -5.7966e-01, -3.3632e-01,
 3.1987e-01, -1.9380e-01,  2.8006e-01,  5.9293e-01,  5.4405e-01,
 3.7746e-01,  1.5074e-01,  1.3050e-01,  1.8917e-01, -8.2223e-03,
-3.1581e-01,  3.0006e-01,  2.2615e-01, -4.7198e-01,  2.9255e-01,
 3.9098e-01,  7.1997e-01, -6.5656e-01, -5.7877e-02, -7.4665e-01,
 4.5998e-01, -1.7215e-01,  3.4801e-01, -3.4560e-01,  3.4529e-01,
 4.9162e-01,  3.4105e-01,  5.2297e-01, -5.5147e-01,  2.9069e-02,
-1.3466e+00,  1.6773e-01, -2.8781e-01, -6.5206e-01, -2.8571e-01,
 4.4820e-01, -8.2600e-01,  2.7299e-01, -1.6052e-01, -4.9491e-01,
-4.4123e-01, -3.0832e-02,  3.5858e-01, -3.2000e-01, -3.6482e-01,
-4.6250e-01,  6.8682e-01,  6.2212e-01, -6.7616e-01,  4.4652e-01,
 6.3600e-02,  5.6310e-01,  9.1847e-01,  4.4499e-01,  4.0972e-01,
 6.3158e-01, -4.8508e-01, -4.0152e-01, -5.0366e-01, -4.5478e-02,
        -4.5590e-01, -7.5448e-01,  6.8782e-01, -3.8171e-01,  2.5105e-01,
         1.3606e-01,  1.9530e-01, -1.6731e-01,  1.8509e-01,  2.4151e-01,
        -1.2497e-01,  7.3939e-02,  8.0915e-02,  4.0503e-01, -4.4470e-01,
         1.4211e-01, -6.2766e-02, -1.0847e+00,  1.5061e-02,  3.1442e-01,
         6.1792e-02, -2.7084e-01, -9.5111e-02,  8.0670e-01,  3.4658e-01,
         1.7020e-01, -3.4658e-01,  6.2307e-01, -3.1217e-01,  6.7439e-02,
        -4.2302e-01,  6.6040e-01,  2.8155e-01, -3.0294e-02,  1.3720e-01,
        -4.0739e-01,  4.6180e-01,  7.8493e-01, -2.0656e-01, -6.3212e-02,
        -8.2126e-01,  1.9584e-01, -8.8633e-02, -1.0430e+00,  2.6545e-01,
         1.2379e+00,  6.7183e-01, -6.8933e-01,  2.3559e-01,  6.4715e-01,
         1.2310e-01, -2.8892e-01,  2.7236e-01, -4.1748e-01,  9.3375e-01,
         1.2763e-01, -4.8758e-01,  4.1386e-01, -5.7510e-01,  3.0247e-02,
        -5.2484e-01, -3.5384e-01, -4.3309e-01, -5.0379e-01, -1.4411e-01,
        -9.1641e-02,  1.3801e-01, -2.2796e-01,  4.0613e-01, -5.3039e-01,
         2.8264e-01,  1.4536e-01, -3.8260e-01, -1.0636e-01, -1.9395e-01,
         2.4888e-01,  3.9480e-01, -1.9706e-01, -9.8293e-02, -5.3219e-01,
         3.9783e-01, -3.8129e-01,  1.8971e-01, -3.3756e-01, -6.2739e-01,
        -7.0887e-01,  3.7720e-01,  2.1101e-01, -5.8501e-01, -4.8129e-01,
         2.2947e-01, -1.3340e-01,  1.1180e+00,  6.6438e-02,  4.8484e-01,
        -2.2995e-01, -3.2569e-01,  4.2933e-02,  3.4589e-01, -2.4830e-01,
         1.0948e-01, -1.3069e-01, -5.7944e-01, -1.9196e-01, -7.8060e-02,
         1.1934e-01, -2.3422e-01, -2.1563e-01, -3.3932e-01,  8.7396e-02,
         2.8107e-01, -4.7970e-01,  2.1479e-01, -1.1446e-01, -5.0093e-01,
        -1.2180e-01,  5.6778e-01, -1.7505e-01,  5.0499e-01, -6.7217e-01,
         3.6737e-01, -1.8358e-01,  2.3660e-01,  2.9736e-02, -3.5624e-01,
        -1.1451e-01,  6.5315e-01, -4.0913e-01, -5.6405e-01,  7.2596e-01,
        -2.3398e-01,  1.7279e-01, -4.2138e-01,  1.0075e-01,  1.4012e-01,
         1.6682e-01,  2.0844e-01,  7.5970e-02,  2.4352e-01,  1.3455e-01,
         1.4141e-01,  1.7959e-01,  1.4143e+00,  1.2107e-01,  3.6410e-01,
        -5.9567e-01,  4.5288e-02,  4.0947e-02, -6.3150e-03, -1.5476e-02,
        -5.4967e-02,  4.0418e-02,  5.3073e-01,  3.6719e-01,  5.9932e-03,
        -6.0457e-01, -4.0461e-02,  3.0526e-01,  2.2857e-01,  3.1699e-01,
        -6.1115e-01, -3.5432e-01, -1.7680e-01,  3.0425e-01,  4.4631e-01,
        -8.0750e-02,  3.3443e-02, -1.7067e-01,  2.9058e-01,  1.1439e-01,
        -1.2962e-02,  2.6377e-01, -2.9394e-01, -2.3483e-01,  2.1922e-01,
         7.9876e-01,  9.3168e-01, -1.6078e-01, -6.8043e-02, -2.8835e-01,
        -3.7442e-02,  8.7288e-01, -3.0851e-02, -2.6066e-01,  8.7559e-01,
         5.2805e-02,  4.2036e-01, -4.4657e-01, -6.0128e-01,  2.4042e-01,
         1.0409e+00,  4.5662e-02, -5.9405e-01, -7.7239e-02,  4.4467e-01,
         2.8607e-01,  3.1795e-01,  4.0720e-01, -4.4101e-01,  1.0156e-01,
         8.4664e-02,  2.2385e-01,  3.2971e-02,  8.5590e-01, -8.3133e-02,
         2.4421e-01, -2.5320e-01,  3.0961e-01,  3.2430e-01,  7.6288e-01,
        -4.1389e-01, -7.4261e-01, -1.0888e-01,  3.0744e-02, -8.4961e-02,
        -3.3006e-01, -9.4410e-02, -1.7684e-01,  9.1215e-02,  3.9247e-01,
        -1.6165e-01, -2.5071e-01,  3.2119e-01, -5.8943e-01,  5.3371e-01,
        -3.9494e-01,  4.5296e-01, -3.2192e-02,  2.9046e-02,  9.0094e-02,
         8.0394e-02, -3.8424e-01, -2.4358e-01, -5.8893e-01, -1.6014e-01,
        -5.0848e-01, -2.1314e-02, -5.0818e-01, -9.5969e-02,  4.0968e-01,
         3.0001e-01, -5.3867e-01,  2.3648e-01, -5.9909e-01,  1.8138e-01,
         5.6676e-01,  3.3343e-01,  3.6125e-02, -7.1760e-01, -2.8207e-01,
        -1.7438e-01, -9.5262e-02,  2.1486e-01, -5.2020e-02, -1.4042e-04,
        -1.5751e-01, -6.5503e-02,  3.1234e-01, -1.2235e-01, -1.2310e-01,
        -2.6430e-01,  6.9013e-01, -8.8553e-01,  1.4780e-01, -4.3434e-01,
         9.2152e-03, -5.1354e-01, -4.0582e-01,  4.0951e-02, -9.0218e-01,
         1.7678e-01,  8.2160e-01,  9.0611e-02,  1.0791e-01, -5.0325e-01,
         5.2987e-01, -4.0699e-01,  1.9170e-01, -2.1431e-01,  4.5255e-01,
         5.6901e-01,  2.7348e-01, -4.4278e-01,  1.8978e-01,  1.3639e-03,
         2.6983e-01,  6.8033e-01, -4.7854e-01,  8.3501e-01,  5.7284e-01,
         5.5622e-01, -3.8540e-01, -1.7700e-02,  3.0426e-01,  2.9132e-01,
        -2.5081e-01, -5.4555e-01,  4.0296e-02, -3.4137e-01,  4.2379e-01,
        -5.2939e-01,  1.7819e-01, -1.2489e-02,  5.5839e-01,  1.0675e-01,
         3.1746e-01, -2.8264e-01, -2.1325e-01], device='cuda:0'))
```

In [ ]:
```python
similarity = nn.CosineSimilarity(dim=0, eps=1e-6)
similarity(bert_watch1, bert_watch2)
```

Out[ ]:
```
tensor(0.3504, device='cuda:0')
```

Document Embeddings

In [ ]:
```python
documents = [Sentence(watch1), Sentence(watch2)]
```

```
In [ ]: from flair.embeddings import TransformerDocumentEmbeddings
        bert_doc = TransformerDocumentEmbeddings('bert-base-uncased')
        bert_doc.embed(documents)
```

```
Out[ ]: [Sentence[58]: " The Hatter was the first to break the silence. `What day of the month is it?' he said, turnin
        g to Alice:  he had taken his watch out of his pocket, and was looking at it uneasily, shaking it every now an
        d then, and holding it to his ear.",
         Sentence[48]: " Alice thought this a very curious thing, and she went nearer to watch them, and just as she c
        ame up to them she heard one of them say, `Look out now, Five!  Don't go splashing paint over me like that!"]
```

In [ ]:
```
documents[0].embedding
```

```
Out[ ]: tensor([-6.4245e-02,  3.5365e-01, -2.4962e-01, -5.3912e-01, -1.9917e-01,
                -2.7712e-01,  1.6942e-01,  1.0867e-01, -4.0799e-01, -1.4945e-01,
                 4.4678e-02, -1.9687e-01, -3.0574e-01,  9.6215e-02,  2.8053e-01,
                 9.0462e-01, -3.0250e-01,  1.7854e-01, -1.6856e-01, -1.7227e-01,
                -1.8270e-01,  4.5121e-01,  3.6129e-01,  1.2309e-01, -1.2047e-02,
                -5.0255e-01,  4.2645e-01, -3.7184e-02, -1.7052e-01, -4.8920e-02,
                 1.0503e-01,  3.3456e-01, -3.8483e-02, -4.7287e-01, -7.4431e-02,
                 2.3990e-01, -8.3232e-02,  1.7974e-01,  4.8939e-01,  9.3026e-03,
                -4.2722e-01,  2.6337e-01, -1.3217e-01, -1.3950e-01, -1.1859e-01,
                 4.5835e-02, -3.9031e+00, -1.0726e-01,  5.1103e-02, -9.9710e-02,
                 1.5913e-01, -9.5331e-02,  2.1513e-01,  4.7707e-01,  2.1226e-01,
                 4.3450e-01, -4.5154e-01,  1.8344e-01, -1.0171e-01, -3.6373e-01,
                 8.9560e-01,  3.5931e-01, -2.8519e-01,  2.9427e-02,  3.3968e-01,
                 4.1062e-01, -2.1051e-02,  2.7801e-02,  2.8888e-01,  4.5761e-01,
                -3.8254e-01, -1.9211e-01, -2.6551e-01,  1.7064e-01, -2.0920e-01,
                -3.3608e-01, -1.0044e-01, -2.9611e-01, -5.6573e-01,  4.3841e-01,
                -8.9239e-02, -1.4645e-02,  2.2009e-01,  2.5755e-01, -2.6711e-01,
                -1.4440e-01, -2.8944e-01,  2.2547e-01,  6.0303e-01,  1.0250e-01,
                -2.0606e-01, -1.5392e-01, -3.2987e-03,  7.7605e-02,  1.8428e-01,
                 1.6114e-01,  2.5668e-01, -8.3616e-02,  2.7078e-01, -5.4946e-02,
                 4.9633e-01, -2.7734e-01,  4.1486e-01, -5.3346e-01, -7.8131e-01,
                 1.7047e-01,  2.8687e-01,  4.9673e-02,  4.6922e-01, -1.6500e+00,
                 4.6991e-01, -1.5328e-01,  2.7729e-01, -1.0157e+00,  1.5022e-01,
                -1.7466e-01,  4.1232e-02, -4.4218e-01, -5.0299e-01,  1.1937e-01,
                 1.8505e-01,  1.7673e-01,  2.1935e-01, -1.3052e+00, -3.4139e-01,
                -3.8845e-02, -2.4191e-01,  2.5905e-01,  1.2551e-01, -1.7338e-02,
                 4.4672e-01,  3.3219e-01, -2.1076e-01, -2.6042e-01,  2.1485e-01,
                 1.6804e-01, -1.6398e-01,  1.1147e-01, -1.4731e-01,  1.0370e-01,
                -2.0025e-01, -1.7985e-01, -2.5822e+00, -9.3411e-02,  5.8352e-01,
                 4.2851e-01,  1.3719e-01, -4.4423e-01, -2.4915e-03,  6.3892e-01,
                 4.1029e-01, -3.2187e-01,  5.8879e-02, -2.1344e-01, -4.0472e-02,
                -4.0414e-01,  3.8056e-01,  1.7888e-01, -4.1893e-01,  1.6880e-01,
                -4.6562e-02, -3.3560e-01,  1.3598e-02,  2.4541e-01, -1.9910e-01,
                 5.3978e-01,  5.7400e-01,  5.7995e-01, -3.8400e-02, -4.4134e-02,
                 3.8312e-01, -1.6357e-02,  7.0812e-01, -1.6653e-01, -4.0108e-01,
                 8.2582e-02,  4.2481e-01, -1.2611e-01,  4.2772e-01, -7.9697e-02,
                -7.4779e-01,  7.4372e-02,  2.0130e-01,  8.0436e-01,  8.2707e-01,
                 3.8594e-01, -2.6690e-02, -2.8838e-01,  2.1367e-01,  2.3883e-01,
                 1.4772e-01, -1.7737e-01, -1.6458e-01, -4.7632e-01, -9.9866e-02,
                -7.7609e-02,  3.6381e-01, -5.1024e-01, -3.6723e-01, -2.4667e-01,
                -2.0106e-01,  1.8970e-01,  1.2936e-01, -7.0922e-01,  1.8065e-01,
                 4.2441e+00,  3.2519e-01, -3.5622e-01,  7.9891e-02,  4.0457e-01,
                -5.2027e-01, -4.0209e-01,  1.1543e-01,  5.4013e-01, -8.5186e-01,
                 5.7481e-01,  3.0116e-01,  3.0603e-01,  3.0003e-02, -3.7785e-01,
                -2.3895e-01,  1.0855e-01, -7.3348e-02, -2.2568e-02, -1.8636e-01,
                 5.5137e-01, -2.5551e-01,  5.6108e-01,  8.9961e-02, -1.4174e+00,
                 1.9548e-01,  1.8128e-01,  5.7270e-01,  3.7286e-01,  2.1192e-02,
                 2.0576e-01, -2.7981e-01, -4.0403e-02, -5.4554e-02, -2.4785e-01,
                -2.4466e-01, -4.0418e-03,  8.9291e-02,  5.5506e-01, -1.7438e-01,
                 4.4969e-01,  3.9400e-01, -5.3704e-01,  6.6374e-01,  3.8592e-01,
                 6.9357e-01,  6.0498e-01, -2.1726e-01, -1.0124e-01, -2.2627e-01,
                 5.7905e-01,  4.2297e-01, -1.2389e-01,  2.0031e-01,  4.5129e-01,
                 1.7262e-01,  1.0963e-02,  3.6436e-01,  5.7858e-01, -5.3210e-01,
                -1.4753e-02,  6.1678e-01, -3.0254e-01, -3.7331e-01,  4.2933e-02,
                -2.8411e-01,  2.5481e-01, -3.1135e-01, -2.4889e+00,  1.2208e-01,
                -8.3915e-02, -1.0063e-01,  1.6028e-01,  9.8308e-02, -1.5816e-01,
                -3.4995e-02, -1.4911e-01, -8.0463e-01,  5.4030e-01,  7.1562e-02,
                 1.7398e-01,  1.4491e-01, -7.0773e-01,  3.9191e-01,  1.0117e-01,
                -2.8370e-01,  2.8936e-01,  1.5133e-01,  5.0763e-02, -2.4023e-01,
                 4.4110e-01, -3.9492e-02,  2.6435e-01,  3.1356e-01, -1.8297e-01,
                -2.4273e-01, -1.7542e-01,  1.4501e-01,  3.0055e-01, -2.8845e-01,
                -1.5901e-01,  1.7493e-01, -2.9559e-01, -3.4490e+00,  1.3099e-01,
                -3.8233e-01, -1.0336e-01,  6.8291e-01,  6.9417e-01,  6.1574e-01,
                -8.2423e-02, -1.2499e-01, -1.2243e-01,  2.8974e-01, -4.1564e-01,
                 6.8800e-02,  6.7222e-01,  5.1716e-01, -3.0253e-02,  3.6390e-01,
                -5.9201e-01,  2.2436e-01, -2.7201e-01, -8.8383e-02, -5.5831e-01,
                 1.0460e-01, -3.3686e-01,  4.6233e-01,  6.3347e-01, -5.7720e-01,
                -4.1045e-01, -6.3033e-02, -2.6181e-01,  2.5220e-01, -5.7010e-01,
                -4.4242e-02, -2.5707e-03, -2.6010e-01, -2.0902e-01,  2.1671e-01,
                -1.5338e-01,  2.3135e-02,  8.3759e-02,  2.4785e-01,  8.1829e-01,
                 4.6562e-01,  1.6161e-01,  1.5305e-01, -2.2000e-01, -2.5913e-01,
                -3.0355e-01,  2.0841e-01, -8.6140e-02, -1.0182e-01,  7.8481e-01,
                 1.0315e+00, -5.8092e-01, -2.5592e-01,  1.4235e-01,  2.2627e-01,
                -3.8135e-01, -1.7177e-01,  6.8986e-01,  9.5218e-01, -4.2518e-01,
                 2.9856e-01, -7.7871e-01,  4.6342e-01,  3.8082e-01,  3.3461e-01,
                -3.6359e-01,  1.3449e-01,  6.0971e-02, -6.3682e-01,  6.5791e-01,
                -3.4900e-01, -1.6919e+00, -1.1290e-01, -8.9324e-03,  7.0164e-01,
                -2.4298e-01, -5.0133e-01, -9.4679e-02,  9.2998e-02, -6.9514e-01,
                 1.6732e-01,  1.0738e-01, -3.5135e-01, -4.1886e-01, -5.6835e-02,
                 1.8975e-01, -6.7744e-01, -5.0675e-02, -2.8237e-01, -9.4651e-02,
                 2.3333e-01,  2.1024e-01, -1.3943e-01,  4.7849e-01, -3.5598e-02,
                -5.0288e-02,  2.2644e-01,  1.7870e-01, -8.6450e-02, -4.6139e-01,
                -9.7895e-02,  9.5692e-03, -6.0332e-01, -1.0339e-01, -6.9934e-01,
                 8.6564e-01,  2.4269e-01,  1.2407e-01, -4.4824e-01,  6.2005e-02,
                 1.1447e-01, -4.3741e-02,  1.5308e+00, -4.7716e-01,  4.9675e-01,
                 8.2663e-01,  4.4054e-01, -8.6615e-02,  2.1151e-01, -1.6980e-02,
                 3.6449e-02,  5.2057e-02, -4.3879e-01, -2.0526e-01, -1.0826e-01,
                -2.0654e-01,  5.4324e-02, -2.3006e-01, -2.3910e-01,  3.5886e-01,
                -3.6865e-01, -4.1345e-01,  3.2852e-01, -2.1309e-01, -7.7967e-01,
                -4.0635e-01,  4.1312e-01,  4.5644e-01,  5.6297e-01, -4.0113e-02,
                -7.8324e-02,  3.5196e-01, -4.0508e-01,  4.9119e-01,  2.8871e-01,
                 1.2004e-02,  2.1122e-02,  5.8201e-01,  7.2554e-02, -1.2406e-01,
                -2.2747e-01, -3.6875e-01,  1.2822e-01, -1.3219e-02, -2.5432e-01,
                -5.4382e-01, -1.6104e-01, -4.9271e-01, -2.7153e-05, -1.6866e-01,
                -2.0367e+00, -1.8150e-01,  7.2432e-01,  2.6644e-01,  2.3441e-01,
                 1.3906e-01, -7.0422e-03,  4.3320e-01, -3.9089e-01,  3.8913e-01,
                -1.3505e-01,  1.8608e-01, -2.7714e-01,  7.7001e-02, -3.5178e-01,
                 3.1395e-01, -6.1126e-01, -1.9384e-01,  3.4077e-02,  7.6535e-02,
                -6.3849e-01,  9.1558e-02,  6.1446e-01, -9.3747e-02,  2.7309e-02,
                -2.9999e-01,  8.1415e-02,  2.3125e-01,  4.6290e-02,  1.2134e-01,
                 1.3592e-01, -5.6097e-01, -2.1794e-01, -6.8162e-01,  6.4019e-01,
                -1.8568e-01, -1.6626e-01,  3.4131e-01,  1.8143e-01,  1.4511e-01,
                 4.1716e-02,  3.5290e-01,  2.6542e-01, -3.3044e-01,  6.5737e-01,
                -4.8462e-01, -4.2369e-01,  5.4556e-01, -4.1632e-01, -7.4869e-02,
                -2.7194e-01, -5.6729e-02, -1.9846e-01, -2.5328e-01, -8.4373e-01,
                -2.5077e-01, -7.6245e-02,  1.3954e-01, -7.3810e-01, -7.1694e-02,
                 3.7180e-01, -6.6641e-01, -7.3948e-02,  1.0740e-02,  6.6874e-02,
                -5.4790e-01, -2.3494e-01, -5.4339e-02, -2.7594e-01,  2.3641e-01,
                 9.0829e-02, -2.4923e-02, -6.5618e-01, -1.5407e-01,  6.5148e-03,
                -1.8896e-01, -4.8536e-01,  4.1967e-01,  4.9990e-01, -2.3470e-01,
                 2.3999e-01,  2.2489e-02,  7.1899e-02,  3.7770e-01,  2.2666e-01,
                -4.6463e-02,  7.9462e-02, -1.3600e-01,  2.6624e-01, -1.0557e-01,
                -5.1542e-01, -2.1026e-01, -3.6529e-01,  3.7681e-01,  4.2558e-01,
                -2.0674e-01,  1.7708e-01, -9.5000e-01,  3.5931e-01, -6.9485e-01,
                 2.6662e-01, -3.8393e-01, -1.4876e-01,  3.5606e-01, -1.9180e-01,
                 5.2490e-02,  6.9162e-01,  1.2743e-01, -2.9465e-01, -6.5748e-01,
                -1.3319e-01, -1.6700e-01,  3.4043e-01,  1.0655e-01,  1.4711e-01,
                -5.7975e-01, -1.9442e-01, -5.5820e-01,  1.5113e+00,  2.5854e-01,
                -7.7182e-02,  1.9113e-01, -2.0155e-01,  5.2039e-02, -6.6809e-02,
                 7.7022e-01,  4.3338e-01,  2.1713e-01, -1.9559e-01,  1.1622e-01,
                 6.9701e-02,  1.9887e-01,  4.4560e-01, -2.0377e-01,  9.3891e-02,
                -3.8858e-01, -1.0287e+00, -8.7257e-02,  3.0796e-01,  1.1579e-01,
                 2.2186e-01,  2.4396e-02, -2.2758e-01,  7.7906e-01,  8.9298e-02,
                -5.7492e-01, -2.8990e-01,  5.5056e-01, -3.1180e-01,  5.1760e-01,
                -2.1881e-01,  3.5772e-01, -9.8653e-01,  3.4400e-02, -1.5934e-01,
                -2.6761e-01,  4.0133e-02,  7.4575e-02,  3.0935e-01,  2.3991e-02,
                -2.4070e-01,  1.9046e-01, -3.6814e-01,  3.0127e-01,  1.0484e-01,
                 3.0938e-02,  3.2499e-01,  4.0934e-01, -1.7192e-01,  5.4189e-01,
                 3.5965e-01,  1.4977e-01, -2.0299e-01,  2.5858e-02, -2.3124e-01,
                -5.2681e-01, -5.0653e-02, -4.0077e-01,  3.5185e-02,  6.1388e-01,
                 1.7713e-01, -5.8146e-01,  1.2276e-01,  2.9924e-02, -5.3960e-02,
                -4.8978e-01,  8.5425e-02, -1.6308e-01,  2.2881e-02,  2.6261e-01,
                 2.0515e-01,  3.4557e-01,  2.7707e-01,  3.1900e-01,  6.0790e-01,
                -5.5657e-01, -4.6023e-01, -1.5954e+00,  1.5275e-01,  3.5691e-01,
                 4.9637e-01, -1.7432e-01,  1.8391e-01,  5.2575e-01,  1.6172e-01,
                -1.6158e-01,  1.2285e-01,  1.7329e-01,  1.0693e+00,  4.1782e-01,
                 2.6315e-01, -2.6413e-02, -3.4535e-01,  3.1312e-01,  3.1309e-03,
                -1.3553e-01,  9.2502e-02,  1.8213e-01,  1.0432e-01, -5.5065e-02,
                -5.2151e-01, -5.9766e-01, -3.0186e-01,  1.1254e-02,  6.2411e-01,
                -2.4277e-01,  2.0318e-01, -1.6011e-01,  8.6397e-01, -2.2691e-02,
                -4.4046e-01, -6.2857e-02, -3.9069e-01,  1.0784e-01,  1.1981e-01,
                 4.9630e-01, -3.8379e-03, -4.2250e-01,  1.0451e+00, -2.6349e-01,
                -1.4386e-01,  2.8545e-01,  2.6442e-01,  5.0848e-01, -3.4545e-01,
                 2.0938e-01,  3.9765e-01, -2.8668e-01, -2.9289e-01,  1.0622e-01,
                 1.2800e-01,  4.2874e-01, -7.5396e-02,  2.4798e-01,  8.1016e-02,
                -2.1906e-02, -2.7718e-01, -7.3759e-02, -5.9759e-02, -1.1127e-01,
                -1.2804e-01,  2.8441e-01, -1.5104e-01, -2.0100e-01,  7.6934e-01,
                 3.0968e-01, -5.9233e-01, -3.4167e-01, -1.5913e-01, -5.3265e-02,
                 2.1583e-01, -3.5234e-01, -1.2197e-01,  3.0294e-01,  7.3882e-01,
                 1.3297e-01, -3.4018e-01, -8.1062e-02, -2.5973e-01,  1.7507e-01,
                 2.3974e-02,  4.1746e-01, -6.2069e+00,  1.1619e-01, -5.7523e-01,
                -2.6148e-01, -3.7661e-01, -2.8330e-01,  1.3297e-01,  3.6957e-01,
                 7.4661e-02, -3.4777e-01,  1.5740e-01,  3.4407e-01, -5.0272e-01,
                 1.7432e-01,  7.9398e-01,  7.3562e-01], device='cuda:0')
```

```
In [ ]: documents[0].tokens[31].embedding
```

```
Out[ ]: tensor([], device='cuda:0')
```

In [ ]:
```python
def get_embeddings(embeddings, sentence):
    sent = Sentence(sentence)
    embeddings.embed(sent)
    if len(sent.embedding):
        return sent.embedding.float()
    else:
        return torch.stack([token.embedding for token in sent.tokens]).float()
```

```
In [ ]: get_embeddings(bert_doc, watch1)
```

```
Out[ ]: tensor([-6.4245e-02,  3.5365e-01, -2.4962e-01, -5.3912e-01, -1.9917e-01,
                -2.7712e-01,  1.6942e-01,  1.0867e-01, -4.0799e-01, -1.4945e-01,
                 4.4679e-02, -1.9687e-01, -3.0574e-01,  9.6215e-02,  2.8053e-01,
                 9.0462e-01, -3.0250e-01,  1.7854e-01, -1.6856e-01, -1.7227e-01,
                -1.8270e-01,  4.5121e-01,  3.6128e-01,  1.2309e-01, -1.2047e-02,
                -5.0255e-01,  4.2645e-01, -3.7184e-02, -1.7052e-01, -4.8920e-02,
                 1.0503e-01,  3.3456e-01, -3.8483e-02, -4.7287e-01, -7.4431e-02,
                 2.3990e-01, -8.3232e-02,  1.7974e-01,  4.8939e-01,  9.3027e-03,
                -4.2722e-01,  2.6337e-01, -1.3217e-01, -1.3950e-01, -1.1859e-01,
                 4.5834e-02, -3.9031e+00, -1.0726e-01,  5.1103e-02, -9.9710e-02,
                 1.5913e-01, -9.5331e-02,  2.1513e-01,  4.7707e-01,  2.1226e-01,
                 4.3450e-01, -4.5154e-01,  1.8344e-01, -1.0171e-01, -3.6373e-01,
                 8.9560e-01,  3.5931e-01, -2.8519e-01,  2.9427e-02,  3.3968e-01,
                 4.1062e-01, -2.1051e-02,  2.7801e-02,  2.8888e-01,  4.5761e-01,
                -3.8254e-01, -1.9211e-01, -2.6551e-01,  1.7064e-01, -2.0920e-01,
                -3.3608e-01, -1.0044e-01, -2.9611e-01, -5.6573e-01,  4.3841e-01,
                -8.9239e-02, -1.4645e-02,  2.2009e-01,  2.5755e-01, -2.6711e-01,
                -1.4440e-01, -2.8944e-01,  2.2546e-01,  6.0303e-01,  1.0250e-01,
                -2.0606e-01, -1.5392e-01, -3.2986e-03,  7.7604e-02,  1.8428e-01,
                 1.6114e-01,  2.5668e-01, -8.3616e-02,  2.7078e-01, -5.4946e-02,
                 4.9633e-01, -2.7734e-01,  4.1486e-01, -5.3346e-01, -7.8131e-01,
                 1.7047e-01,  2.8687e-01,  4.9674e-02,  4.6922e-01, -1.6500e+00,
                 4.6991e-01, -1.5328e-01,  2.7729e-01, -1.0157e+00,  1.5022e-01,
                -1.7466e-01,  4.1232e-02, -4.4218e-01, -5.0299e-01,  1.1937e-01,
                 1.8505e-01,  1.7673e-01,  2.1935e-01, -1.3052e+00, -3.4139e-01,
                -3.8845e-02, -2.4191e-01,  2.5906e-01,  1.2551e-01, -1.7338e-02,
                 4.4672e-01,  3.3219e-01, -2.1076e-01, -2.6042e-01,  2.1485e-01,
                 1.6804e-01, -1.6398e-01,  1.1147e-01, -1.4731e-01,  1.0370e-01,
                -2.0025e-01, -1.7985e-01, -2.5822e+00, -9.3411e-02,  5.8352e-01,
                 4.2851e-01,  1.3719e-01, -4.4423e-01, -2.4913e-03,  6.3892e-01,
                 4.1029e-01, -3.2187e-01,  5.8879e-02, -2.1344e-01, -4.0472e-02,
                -4.0414e-01,  3.8056e-01,  1.7888e-01, -4.1893e-01,  1.6880e-01,
                -4.6562e-02, -3.3560e-01,  1.3598e-02,  2.4541e-01, -1.9910e-01,
                 5.3978e-01,  5.7400e-01,  5.7995e-01, -3.8400e-02, -4.4135e-02,
                 3.8312e-01, -1.6357e-02,  7.0812e-01, -1.6653e-01, -4.0108e-01,
                 8.2581e-02,  4.2481e-01, -1.2611e-01,  4.2772e-01, -7.9696e-02,
                -7.4779e-01,  7.4372e-02,  2.0130e-01,  8.0436e-01,  8.2707e-01,
                 3.8594e-01, -2.6690e-02, -2.8838e-01,  2.1367e-01,  2.3883e-01,
                 1.4773e-01, -1.7737e-01, -1.6458e-01, -4.7632e-01, -9.9866e-02,
                -7.7609e-02,  3.6381e-01, -5.1024e-01, -3.6723e-01, -2.4667e-01,
                -2.0106e-01,  1.8970e-01,  1.2936e-01, -7.0922e-01,  1.8065e-01,
                 4.2441e+00,  3.2519e-01, -3.5622e-01,  7.9890e-02,  4.0457e-01,
                -5.2027e-01, -4.0209e-01,  1.1543e-01,  5.4013e-01, -8.5187e-01,
                 5.7481e-01,  3.0116e-01,  3.0603e-01,  3.0002e-02, -3.7785e-01,
                -2.3895e-01,  1.0855e-01, -7.3348e-02, -2.2569e-02, -1.8636e-01,
                 5.5137e-01, -2.5551e-01,  5.6109e-01,  8.9961e-02, -1.4174e+00,
                 1.9548e-01,  1.8128e-01,  5.7270e-01,  3.7286e-01,  2.1192e-02,
                 2.0576e-01, -2.7981e-01, -4.0403e-02, -5.4554e-02, -2.4785e-01,
                -2.4466e-01, -4.0415e-03,  8.9292e-02,  5.5506e-01, -1.7438e-01,
                 4.4969e-01,  3.9400e-01, -5.3704e-01,  6.6374e-01,  3.8592e-01,
                 6.9357e-01,  6.0498e-01, -2.1726e-01, -1.0124e-01, -2.2627e-01,
                 5.7905e-01,  4.2297e-01, -1.2389e-01,  2.0031e-01,  4.5129e-01,
                 1.7262e-01,  1.0963e-02,  3.6436e-01,  5.7858e-01, -5.3210e-01,
                -1.4754e-02,  6.1678e-01, -3.0254e-01, -3.7331e-01,  4.2933e-02,
                -2.8411e-01,  2.5481e-01, -3.1135e-01, -2.4889e+00,  1.2208e-01,
                -8.3916e-02, -1.0063e-01,  1.6028e-01,  9.8307e-02, -1.5816e-01,
                -3.4994e-02, -1.4911e-01, -8.0463e-01,  5.4030e-01,  7.1562e-02,
                 1.7398e-01,  1.4491e-01, -7.0773e-01,  3.9191e-01,  1.0117e-01,
                -2.8370e-01,  2.8936e-01,  1.5133e-01,  5.0763e-02, -2.4023e-01,
                 4.4110e-01, -3.9492e-02,  2.6435e-01,  3.1356e-01, -1.8297e-01,
                -2.4273e-01, -1.7542e-01,  1.4501e-01,  3.0055e-01, -2.8845e-01,
                -1.5901e-01,  1.7493e-01, -2.9559e-01, -3.4490e+00,  1.3099e-01,
                -3.8233e-01, -1.0336e-01,  6.8291e-01,  6.9417e-01,  6.1574e-01,
                -8.2423e-02, -1.2499e-01, -1.2243e-01,  2.8974e-01, -4.1564e-01,
                 6.8800e-02,  6.7222e-01,  5.1716e-01, -3.0253e-02,  3.6390e-01,
                -5.9201e-01,  2.2436e-01, -2.7201e-01, -8.8383e-02, -5.5831e-01,
                 1.0460e-01, -3.3686e-01,  4.6233e-01,  6.3347e-01, -5.7720e-01,
                -4.1045e-01, -6.3033e-02, -2.6181e-01,  2.5220e-01, -5.7010e-01,
                -4.4242e-02, -2.5706e-03, -2.6010e-01, -2.0902e-01,  2.1671e-01,
                -1.5338e-01,  2.3135e-02,  8.3758e-02,  2.4785e-01,  8.1829e-01,
                 4.6562e-01,  1.6161e-01,  1.5305e-01, -2.2000e-01, -2.5913e-01,
                -3.0355e-01,  2.0841e-01, -8.6139e-02, -1.0182e-01,  7.8481e-01,
                 1.0315e+00, -5.8092e-01, -2.5592e-01,  1.4235e-01,  2.2627e-01,
                -3.8135e-01, -1.7177e-01,  6.8986e-01,  9.5218e-01, -4.2518e-01,
                 2.9856e-01, -7.7871e-01,  4.6342e-01,  3.8082e-01,  3.3461e-01,
                -3.6359e-01,  1.3449e-01,  6.0971e-02, -6.3682e-01,  6.5791e-01,
                -3.4900e-01, -1.6919e+00, -1.1290e-01, -8.9325e-03,  7.0164e-01,
                -2.4298e-01, -5.0133e-01, -9.4679e-02,  9.2998e-02, -6.9514e-01,
                 1.6732e-01,  1.0738e-01, -3.5135e-01, -4.1886e-01, -5.6835e-02,
                 1.8975e-01, -6.7744e-01, -5.0675e-02, -2.8237e-01, -9.4652e-02,
                 2.3333e-01,  2.1025e-01, -1.3943e-01,  4.7849e-01, -3.5598e-02,
                -5.0288e-02,  2.2644e-01,  1.7870e-01, -8.6450e-02, -4.6139e-01,
                -9.7895e-02,  9.5687e-03, -6.0332e-01, -1.0339e-01, -6.9934e-01,
                 8.6564e-01,  2.4269e-01,  1.2407e-01, -4.4824e-01,  6.2005e-02,
                 1.1447e-01, -4.3741e-02,  1.5308e+00, -4.7716e-01,  4.9675e-01,
                 8.2663e-01,  4.4054e-01, -8.6616e-02,  2.1151e-01, -1.6980e-02,
                 3.6449e-02,  5.2057e-02, -4.3879e-01, -2.0526e-01, -1.0826e-01,
                -2.0654e-01,  5.4324e-02, -2.3006e-01, -2.3910e-01,  3.5886e-01,
                -3.6865e-01, -4.1345e-01,  3.2852e-01, -2.1309e-01, -7.7967e-01,
                -4.0635e-01,  4.1312e-01,  4.5644e-01,  5.6297e-01, -4.0113e-02,
                -7.8324e-02,  3.5196e-01, -4.0508e-01,  4.9119e-01,  2.8871e-01,
                 1.2004e-02,  2.1122e-02,  5.8201e-01,  7.2554e-02, -1.2406e-01,
                -2.2747e-01, -3.6875e-01,  1.2822e-01, -1.3220e-02, -2.5432e-01,
                -5.4382e-01, -1.6104e-01, -4.9272e-01, -2.7074e-05, -1.6866e-01,
                -2.0367e+00, -1.8150e-01,  7.2432e-01,  2.6644e-01,  2.3441e-01,
                 1.3906e-01, -7.0424e-03,  4.3320e-01, -3.9089e-01,  3.8913e-01,
                -1.3505e-01,  1.8608e-01, -2.7714e-01,  7.7001e-02, -3.5178e-01,
                 3.1395e-01, -6.1126e-01, -1.9384e-01,  3.4077e-02,  7.6536e-02,
                -6.3849e-01,  9.1557e-02,  6.1446e-01, -9.3747e-02,  2.7309e-02,
                -2.9999e-01,  8.1415e-02,  2.3125e-01,  4.6289e-02,  1.2134e-01,
                 1.3592e-01, -5.6097e-01, -2.1794e-01, -6.8162e-01,  6.4019e-01,
                -1.8568e-01, -1.6626e-01,  3.4131e-01,  1.8143e-01,  1.4511e-01,
                 4.1716e-02,  3.5290e-01,  2.6542e-01, -3.3044e-01,  6.5737e-01,
                -4.8462e-01, -4.2369e-01,  5.4556e-01, -4.1632e-01, -7.4868e-02,
                -2.7194e-01, -5.6729e-02, -1.9846e-01, -2.5328e-01, -8.4373e-01,
                -2.5077e-01, -7.6245e-02,  1.3954e-01, -7.3810e-01, -7.1693e-02,
                 3.7180e-01, -6.6641e-01, -7.3948e-02,  1.0741e-02,  6.6875e-02,
                -5.4790e-01, -2.3494e-01, -5.4339e-02, -2.7594e-01,  2.3641e-01,
                 9.0829e-02, -2.4924e-02, -6.5618e-01, -1.5407e-01,  6.5148e-03,
                -1.8896e-01, -4.8536e-01,  4.1967e-01,  4.9990e-01, -2.3470e-01,
                 2.3999e-01,  2.2488e-02,  7.1898e-02,  3.7770e-01,  2.2666e-01,
                -4.6463e-02,  7.9462e-02, -1.3600e-01,  2.6624e-01, -1.0557e-01,
                -5.1542e-01, -2.1026e-01, -3.6529e-01,  3.7681e-01,  4.2558e-01,
                -2.0673e-01,  1.7708e-01, -9.5000e-01,  3.5931e-01, -6.9485e-01,
                 2.6662e-01, -3.8393e-01, -1.4876e-01,  3.5606e-01, -1.9180e-01,
                 5.2491e-02,  6.9162e-01,  1.2743e-01, -2.9465e-01, -6.5748e-01,
                -1.3319e-01, -1.6700e-01,  3.4043e-01,  1.0655e-01,  1.4711e-01,
                -5.7975e-01, -1.9442e-01, -5.5820e-01,  1.5113e+00,  2.5854e-01,
                -7.7182e-02,  1.9113e-01, -2.0155e-01,  5.2040e-02, -6.6809e-02,
                 7.7022e-01,  4.3337e-01,  2.1713e-01, -1.9559e-01,  1.1622e-01,
                 6.9701e-02,  1.9887e-01,  4.4560e-01, -2.0377e-01,  9.3891e-02,
                -3.8858e-01, -1.0287e+00, -8.7257e-02,  3.0796e-01,  1.1579e-01,
                 2.2186e-01,  2.4396e-02, -2.2758e-01,  7.7906e-01,  8.9299e-02,
                -5.7492e-01, -2.8990e-01,  5.5056e-01, -3.1180e-01,  5.1760e-01,
                -2.1880e-01,  3.5772e-01, -9.8653e-01,  3.4400e-02, -1.5934e-01,
                -2.6761e-01,  4.0133e-02,  7.4575e-02,  3.0935e-01,  2.3991e-02,
                -2.4070e-01,  1.9046e-01, -3.6814e-01,  3.0126e-01,  1.0484e-01,
                 3.0939e-02,  3.2499e-01,  4.0934e-01, -1.7192e-01,  5.4189e-01,
                 3.5965e-01,  1.4977e-01, -2.0299e-01,  2.5858e-02, -2.3124e-01,
                -5.2681e-01, -5.0653e-02, -4.0077e-01,  3.5185e-02,  6.1388e-01,
                 1.7713e-01, -5.8146e-01,  1.2276e-01,  2.9924e-02, -5.3960e-02,
                -4.8978e-01,  8.5425e-02, -1.6308e-01,  2.2882e-02,  2.6261e-01,
                 2.0515e-01,  3.4557e-01,  2.7707e-01,  3.1900e-01,  6.0790e-01,
                -5.5657e-01, -4.6023e-01, -1.5954e+00,  1.5275e-01,  3.5691e-01,
                 4.9637e-01, -1.7432e-01,  1.8391e-01,  5.2575e-01,  1.6172e-01,
                -1.6158e-01,  1.2285e-01,  1.7329e-01,  1.0693e+00,  4.1782e-01,
                 2.6315e-01, -2.6412e-02, -3.4535e-01,  3.1312e-01,  3.1306e-03,
                -1.3553e-01,  9.2502e-02,  1.8213e-01,  1.0432e-01, -5.5065e-02,
                -5.2151e-01, -5.9766e-01, -3.0186e-01,  1.1254e-02,  6.2411e-01,
                -2.4277e-01,  2.0318e-01, -1.6011e-01,  8.6397e-01, -2.2691e-02,
                -4.4046e-01, -6.2856e-02, -3.9069e-01,  1.0784e-01,  1.1981e-01,
                 4.9630e-01, -3.8379e-03, -4.2250e-01,  1.0451e+00, -2.6349e-01,
                -1.4386e-01,  2.8545e-01,  2.6442e-01,  5.0848e-01, -3.4544e-01,
                 2.0938e-01,  3.9765e-01, -2.8668e-01, -2.9289e-01,  1.0621e-01,
                 1.2800e-01,  4.2874e-01, -7.5396e-02,  2.4798e-01,  8.1016e-02,
                -2.1905e-02, -2.7718e-01, -7.3759e-02, -5.9758e-02, -1.1127e-01,
                -1.2804e-01,  2.8441e-01, -1.5104e-01, -2.0100e-01,  7.6934e-01,
                 3.0968e-01, -5.9233e-01, -3.4167e-01, -1.5913e-01, -5.3265e-02,
                 2.1583e-01, -3.5234e-01, -1.2197e-01,  3.0294e-01,  7.3882e-01,
                 1.3297e-01, -3.4018e-01, -8.1062e-02, -2.5973e-01,  1.7507e-01,
                 2.3974e-02,  4.1746e-01, -6.2069e+00,  1.1619e-01, -5.7523e-01,
                -2.6148e-01, -3.7661e-01, -2.8330e-01,  1.3297e-01,  3.6957e-01,
                 7.4662e-02, -3.4777e-01,  1.5740e-01,  3.4407e-01, -5.0272e-01,
                 1.7432e-01,  7.9398e-01,  7.3562e-01], device='cuda:0')
```

Model III - Preprocessing Embeddings

We need to use the get_embeddings for every sentence, therefore we can use the map function of the HF dataset, then we need the embeddings to be PyTorch Tensors

Data Preparation

```
In [ ]: train_dataset_doc = train_dataset.map(lambda row: {'embeddings': get_embeddings(bert_doc, row['sentence'])})
        test_dataset_doc = test_dataset.map(lambda row: {'embeddings': get_embeddings(bert_doc, row['sentence'])})
```

```
In [ ]: train_dataset_doc.set_format(type='torch', columns=['embeddings', 'labels'])
        test_dataset_doc.set_format(type='torch', columns=['embeddings', 'labels'])
```

```
In [ ]: train_dataset_doc['embeddings']
```

```
Out[ ]: tensor([[ 0.2049, -0.0373, -0.3377,  ..., -0.1574,  0.3196,  0.2615],
                [-0.0672, -0.1841, -0.3303,  ...,  0.1671,  0.5699,  0.5314],
                [ 0.2318,  0.7190,  0.0883,  ...,  0.1930,  0.4507,  0.4888],
                ...,
                [ 0.1397,  0.3430,  0.0459,  ..., -0.1221,  0.3124,  0.7351],
                [ 0.3447,  0.6325,  0.0887,  ...,  0.1721,  0.5595,  0.4038],
                [-0.1259, -0.0285, -0.3400,  ..., -0.0826,  0.4412,  0.2061]])
```

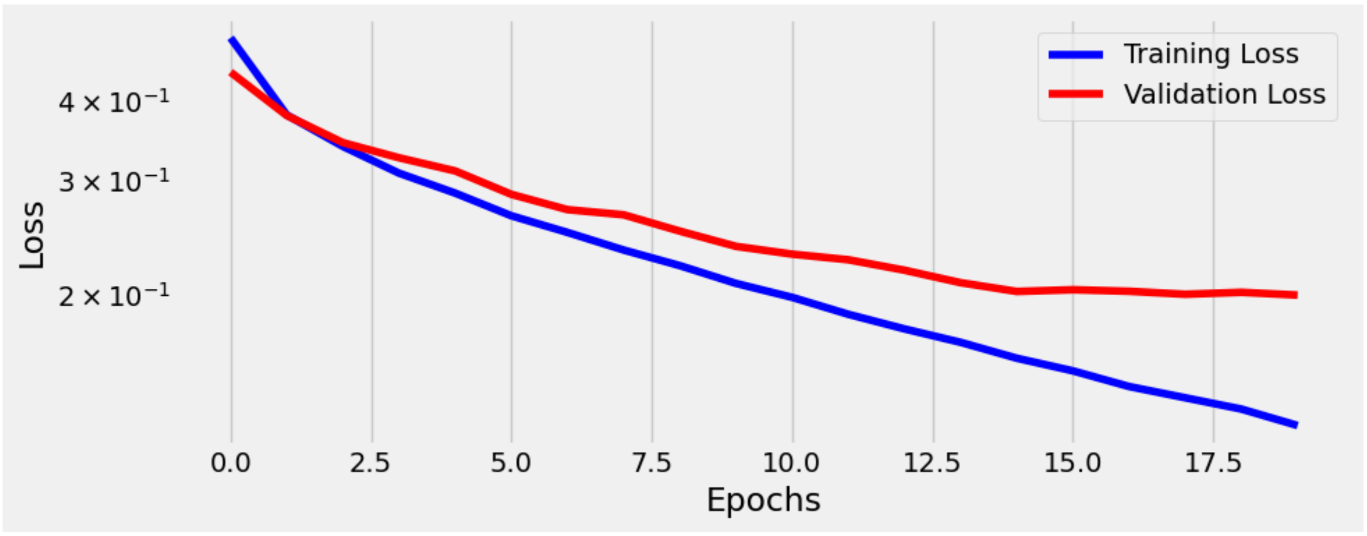
```
In [ ]: train_dataset_doc = TensorDataset(train_dataset_doc['embeddings'].float(),
                                          train_dataset_doc['labels'].view(-1, 1).float())
        generator = torch.Generator()
        train_loader = DataLoader(train_dataset_doc, batch_size=32, shuffle=True, generator=generator)

        test_dataset_doc = TensorDataset(test_dataset_doc['embeddings'].float(),
                                         test_dataset_doc['labels'].view(-1, 1).float())
        test_loader = DataLoader(test_dataset_doc, batch_size=32, shuffle=True)
```

Model Configuration & Training

```
In [ ]: torch.manual_seed(41)
        model = nn.Sequential(
            # Classifier
            nn.Linear(bert_doc.embedding_length, 3),
            nn.ReLU(),
            nn.Linear(3, 1)
        )
        loss_fn = nn.BCEWithLogitsLoss()
        optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

```
In [ ]: sbs_doc_emb = StepByStep(model, loss_fn, optimizer)
        sbs_doc_emb.set_loaders(train_loader, test_loader)
        sbs_doc_emb.train(20)
```

```
In [ ]: fig = sbs_doc_emb.plot_losses()
```



```
In [ ]: StepByStep.loader_apply(test_loader, sbs_doc_emb.correct)
```

```
Out[ ]: tensor([[424, 444],
                [307, 327]])
```

BERT

Bidirectional **E**ncoder **R**epresentation from **T**ransformers $\rightarrow$ **BERT**

It is a model based on a **transformer encoder**.

It was introduced in a paper titled: *BERT: Pre-training of Bidirectional Transformers for Language Understanding* (2019)

Some number to give you an idea: Trained on huge corpora: BookCorpus, 800M of words, 11.038 unpublished books and English Wikipedia with 2.5B of words

12 layers, 12 attention heads, 768 hidden dimensions, with a total of 110 Milion Parameters.

What does this mean? That we don't have -- as personal users -- the computational resources to train such a kind of models.

HuggingFace is at our disposal and there are many different version of BERT available.

What do we want to do now?

USE A PRE-TRAINED VERSION OF BERT, FINE-TUNING IT FOR OUR PURPOSES AND EVALUATE ON OUR SENTENCE CLASSIFIER

In [ ]:
```python
'''
if you want to try different models without having to import their
corresponding classes, you can use HuggingFace's AutoModel

It infers the corret model class based on the name of the model you are loading
'''
from transformers import AutoModel
auto_model = AutoModel.from_pretrained('bert-base-uncased')
print(auto_model.__class__)
```

```
<class 'transformers.models.bert.modeling_bert.BertModel'>
```

In [ ]:
```python
'''
Or you can import the class model
'''

from transformers import BertModel
bert_model = BertModel.from_pretrained('bert-base-uncased')
```

In [ ]:
```python
bert_model.config
```

Out[ ]:
```
BertConfig {
  "_name_or_path": "bert-base-uncased",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.38.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 30522
}
```

We are able to recognize some of these parameters, right? hidden_size, num_attention_heads, num_hidden_layers...

Some of them will be explained in few minutes.

But first of all, our model needs to receive inputs and these inputs need to be **TOKENIZED**

Tokenization

We can consider the tokenization as a pre-processing step, and since we are going to use a pre-trained BERT model, we need to use the same tokenizer that was used during the pre-training.

So, in HF each pre-trained model has its own pre-trained tokenizer as well.

Let's create our BERT tokenizer...

```
In [ ]: from transformers import BertTokenizer
        bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
        len(bert_tokenizer.vocab)

Out[ ]: 30522
```

Only `30522` tokens...

But in reality these are not exactly words but they may also be **word pieces**.

Before, for words not belonging to our vocabulary we used the special token `[UNK]`. This approach gets some information loss, all the unknown words are replaced with the same token.

The approach defined here is a litlle bit different. We disassemble an unknown word into its components, and for instance the word `inexplicably` can be disassembled into five word pieces:

`inexplicably` $\longrightarrow$ `in + ##ex + ##pl + ##ica + ##bly`

Every word pieces is prefixed with `##` to indicate that is doesn't stand on its own as a word.

Therefore, an unknown word becomes a concatenation of **word-pieces**

```
In [ ]: sentence1 = 'Alice is inexplicably following the white rabbit'
        sentence2 = 'Follow the white rabbit, Neo'
        tokens = bert_tokenizer(sentence1, sentence2, return_tensors='pt')
        tokens

Out[ ]: {'input_ids': tensor([[  101,  5650,  2003,  1999, 10288, 24759,  5555,  6321,  2206,  1996,
                 2317, 10442,   102,  3582,  1996,  2317, 10442,  1010,  9253,   102]]), 'token_type_ids': tensor
        ([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

- input_ids contains the token id,
- token_type_ids contains the sentence index
- the attention mask is self explanatory.

```
In [ ]: '''
        We take the ids (input_ids)
        and we convert them to the corresponding word pieces (tokens)
        '''

        print(bert_tokenizer.convert_ids_to_tokens(tokens['input_ids'][0]))

        ['[CLS]', 'alice', 'is', 'in', '##ex', '##pl', '##ica', '##bly', 'following', 'the', 'white', 'rabbit', '[SE
        P]', 'follow', 'the', 'white', 'rabbit', ',', 'neo', '[SEP]']
```

- [CLS] at the start, the classifier token
- [SEP] between the two sentences and at the end
- inexplicably got disassembled into word pieces

In [ ]:
```
'''
As for the model you can use the AutoTokenizer
to try different tokenizers without importing their classes
'''

from transformers import AutoTokenizer
auto_tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
print(auto_tokenizer.__class__)
```

```
<class 'transformers.models.bert.tokenization_bert_fast.BertTokenizerFast'>
```

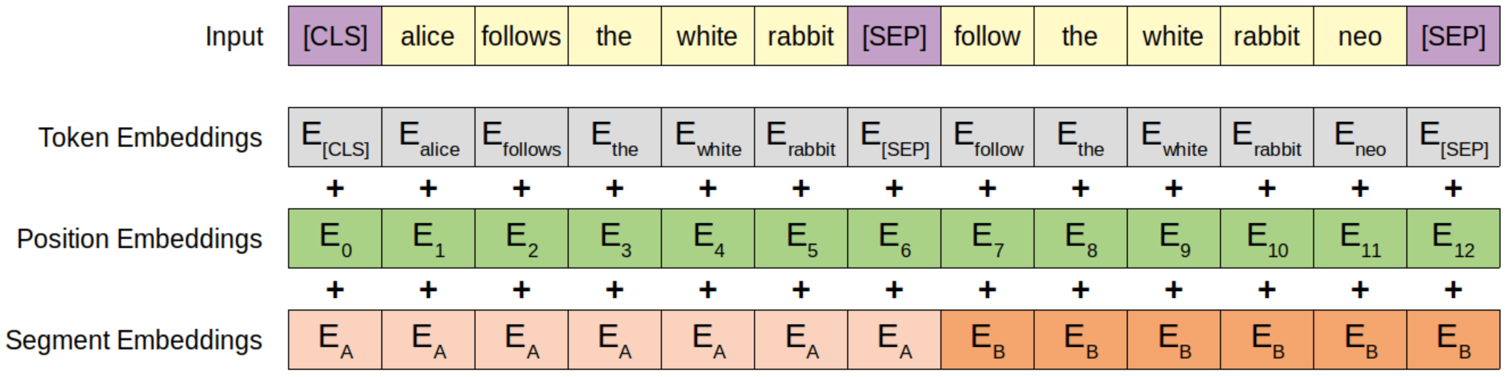
Input Embeddings

Once the sentences are tokenized, we can use their tokens' IDs to look up the corresponding embeddings, as usual.

1. BERT is a transformer encoder, and it needs positional information, and BERT uses **position embeddings** a. position encoding used before had fixed values for each position, the **position embeddings** are learned by the model, as any other embedding layer. The number of entries is defined by the maximum length of the sequence (see parameters above).
2. BERT adds a third embedding: segment embedding, which is a position embedding at the sentence level

Original Design of BERT

- BERT was designed to handle tasks involving one sentence or two sentences.
- For single-sentence tasks (e.g., sentiment analysis), the input is just one sentence.
- For tasks requiring two sentences (e.g., next sentence prediction or sentence-pair classification), the input consists of two segments: Sentence A [SEP] Sentence B.


Input
[CLS]
alice
follows
the
white
rabbit
[SEP]
follow
the
white
rabbit
neo
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{alice} $E_{follows}$ E_{the} E_{white} E_{rabbit} $E_{[SEP]}$ E_{follow} E_{the} E_{white} E_{rabbit} E_{neo} $E_{[SEP]}$
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10} E_{11} E_{12}
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B E_B

In [ ]:
```
input_embeddings = bert_model.embeddings
input_embeddings
```

Out[ ]:
```
BertEmbeddings(
  (word_embeddings): Embedding(30522, 768, padding_idx=0)
  (position_embeddings): Embedding(512, 768)
  (token_type_embeddings): Embedding(2, 768)
  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
```

In [ ]:
```
token_embeddings = input_embeddings.word_embeddings
token_embeddings
```

Out[ ]:
```
Embedding(30522, 768, padding_idx=0)
```

30522 entries, and 768 hidden dimensions

In [ ]:
```
input_token_emb = token_embeddings(tokens['input_ids'])
input_token_emb,input_token_emb.shape
```

Out[ ]:
```
(tensor([[[ 1.3630e-02, -2.6490e-02, -2.3503e-02,  ...,  8.6805e-03,
            7.1340e-03,  1.5147e-02],
          [-6.9710e-02, -8.8202e-02,  5.0619e-03,  ...,  1.4105e-02,
            2.1815e-02, -1.3769e-02],
          [-3.6044e-02, -2.4606e-02, -2.5735e-02,  ...,  3.3691e-03,
           -1.8300e-03,  2.6855e-02],
          ...,
          [ 5.2089e-05, -1.0468e-02, -9.9103e-03,  ...,  1.4558e-02,
            1.3217e-02,  2.2406e-02],
          [-3.5037e-02, -7.2933e-02, -3.6124e-02,  ..., -5.7723e-02,
           -5.5074e-03,  7.2688e-03],
          [-1.4521e-02, -9.9615e-03,  6.0263e-03,  ..., -2.5035e-02,
            4.6379e-03, -1.5378e-03]]], grad_fn=<EmbeddingBackward0>),
 torch.Size([1, 20, 768]))
```

In [ ]:
```
position_embeddings = input_embeddings.position_embeddings
position_embeddings
```

Out[ ]:
```
Embedding(512, 768)
```

In [ ]:
```
position_ids = torch.arange(512).expand((1, -1))
position_ids
```

Out[ ]:
```
tensor([[  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
          14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,  27,
          28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,  40,  41,
          42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,  53,  54,  55,
          56,  57,  58,  59,  60,  61,  62,  63,  64,  65,  66,  67,  68,  69,
          70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  80,  81,  82,  83,
          84,  85,  86,  87,  88,  89,  90,  91,  92,  93,  94,  95,  96,  97,
          98,  99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,
         112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
         126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
         140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153,
         154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167,
         168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
         182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
         196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209,
         210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223,
         224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237,
         238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251,
         252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265,
         266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279,
         280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293,
         294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307,
         308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321,
         322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335,
         336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349,
         350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363,
         364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377,
         378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391,
         392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405,
         406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419,
         420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433,
         434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447,
         448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461,
         462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475,
         476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489,
         490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503,
         504, 505, 506, 507, 508, 509, 510, 511]])
```

```
In [ ]: seq_length = tokens['input_ids'].size(1)
        input_pos_emb = position_embeddings(position_ids[:, :seq_length])
        input_pos_emb,input_pos_emb.shape
```

```
Out[ ]: (tensor([[[ 1.7505e-02, -2.5631e-02, -3.6642e-02,  ...,  3.3437e-05,
                    6.8312e-04,  1.5441e-02],
                  [ 7.7580e-03,  2.2613e-03, -1.9444e-02,  ...,  2.8910e-02,
                    2.9753e-02, -5.3247e-03],
                  [-1.1287e-02, -1.9644e-03, -1.1573e-02,  ...,  1.4908e-02,
                    1.8741e-02, -7.3140e-03],
                  ...,
                  [-9.2809e-03,  8.3268e-03, -4.1643e-03,  ...,  3.4903e-02,
                   -1.8319e-02, -2.9017e-03],
                  [-8.5999e-03,  3.2205e-04, -2.1249e-03,  ...,  2.7744e-02,
                   -7.2760e-03, -2.0280e-03],
                  [-3.4622e-04, -8.3709e-04, -2.2228e-02,  ...,  2.3493e-02,
                   -4.5198e-04, -5.7741e-04]]], grad_fn=<EmbeddingBackward0>),
         torch.Size([1, 20, 768]))
```

```
In [ ]: segment_embeddings = input_embeddings.token_type_embeddings
        segment_embeddings
```

```
Out[ ]: Embedding(2, 768)
```

```
In [ ]: input_seg_emb = segment_embeddings(tokens['token_type_ids'])
        input_seg_emb
```

```
Out[ ]: tensor([[[ 0.0004,  0.0110,  0.0037,  ..., -0.0066, -0.0034, -0.0086],
                 [ 0.0004,  0.0110,  0.0037,  ..., -0.0066, -0.0034, -0.0086],
                 [ 0.0004,  0.0110,  0.0037,  ..., -0.0066, -0.0034, -0.0086],
                 ...,
                 [ 0.0011, -0.0030, -0.0032,  ...,  0.0047, -0.0052, -0.0112],
                 [ 0.0011, -0.0030, -0.0032,  ...,  0.0047, -0.0052, -0.0112],
                 [ 0.0011, -0.0030, -0.0032,  ...,  0.0047, -0.0052, -0.0112]]],
               grad_fn=<EmbeddingBackward0>)
```

BERT adds all three embeddings, then layer normalize and dropout, but these are the inputs that BERT uses

```
In [ ]: input_emb = input_token_emb + input_pos_emb + input_seg_emb
        input_emb
```

```
Out[ ]: tensor([[[ 0.0316, -0.0411, -0.0564,  ...,  0.0021,  0.0044,  0.0219],
                 [-0.0615, -0.0750, -0.0107,  ...,  0.0364,  0.0482, -0.0277],
                 [-0.0469, -0.0156, -0.0336,  ...,  0.0117,  0.0135,  0.0109],
                 ...,
                 [-0.0081, -0.0051, -0.0172,  ...,  0.0542, -0.0103,  0.0083],
                 [-0.0425, -0.0756, -0.0414,  ..., -0.0252, -0.0180, -0.0060],
                 [-0.0138, -0.0138, -0.0194,  ...,  0.0032, -0.0011, -0.0133]]],
               grad_fn=<AddBackward0>)
```

Pretraining Tasks

BERT is a autoencoding model because it is a trasfomer encoder and because it was trained to reconstruct sentences from corrupted inputs.

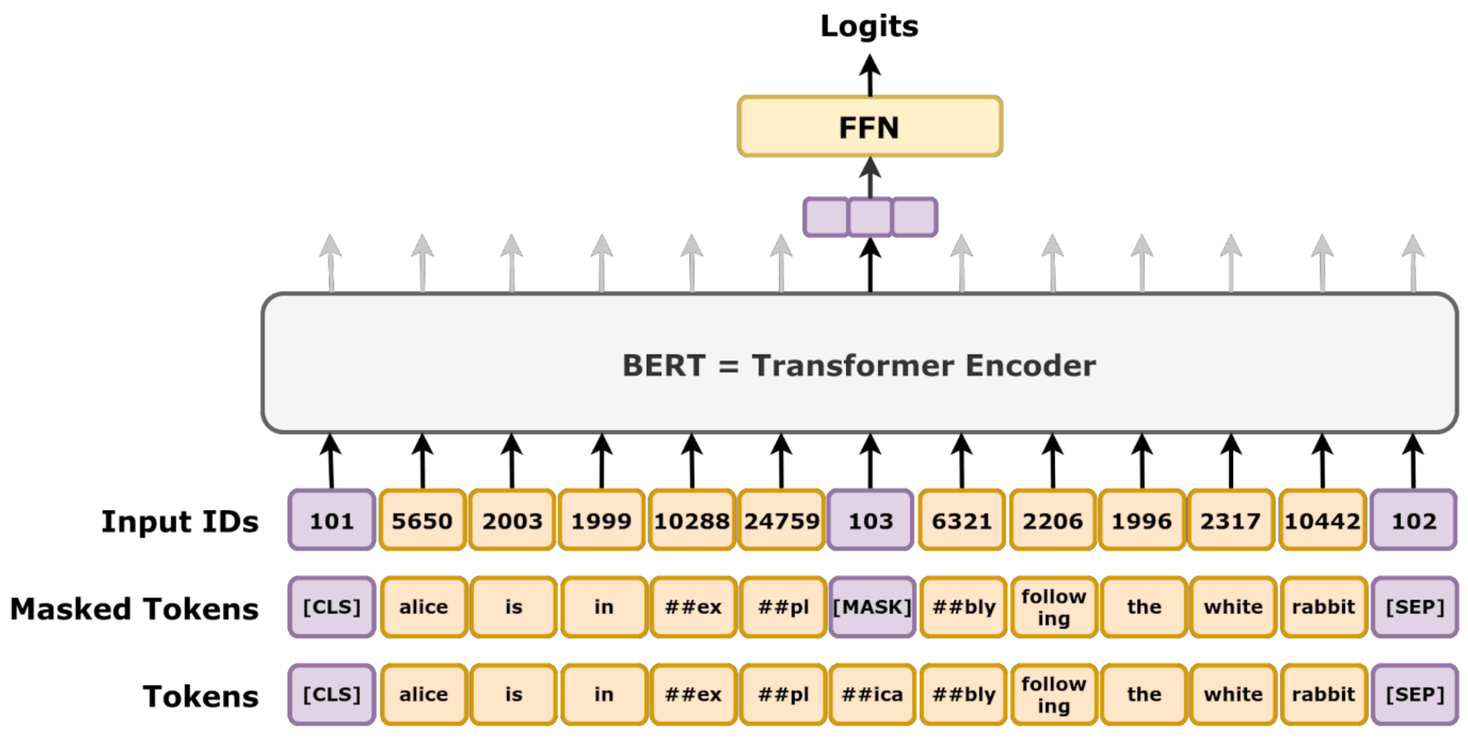
This type of Language models are called masked language models (MLM) pre-training task.

It tries to predict a masked word/token that is inside a sentence, filling the blanks as the continous bag-of-words (CBoW) does.

There are strategies to select which token has to be masked, the target of our encoder is the original sentence.

In particular, BERT computes the logits only for the randomly masked inputs, the others are not used to compute the loss.

Masked Language Model (MLM)



In [ ]:
```python
sentence = 'Alice is inexplicably following the white rabbit'
tokens = bert_tokenizer(sentence)
tokens['input_ids']
```

Out[ ]:
```
[101, 5650, 2003, 1999, 10288, 24759, 5555, 6321, 2206, 1996, 2317, 10442, 102]
```

To do the masking we can use the `DataCollatorForLanguageModeling` class tha does the work for us.

In [ ]:
```python
from transformers import DataCollatorForLanguageModeling
torch.manual_seed(41)
data_collator = DataCollatorForLanguageModeling(tokenizer=bert_tokenizer, mlm_probability=0.15)
mlm_tokens = data_collator([tokens])
mlm_tokens
```

Out[ ]:
```
{'input_ids': tensor([[  101,  5650,  2003,  1999, 10288, 24759,   103,  6321,  2206,  1996,
          2317, 10442,   102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]), 'attenti
on_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]), 'labels': tensor([[-100, -100, -100, -100, -100,
-100, 5555, -100, -100, -100, -100, -100,
         -100]])}
```

`python from transformers import DataCollatorForLanguageModeling`

This is a utility from HuggingFace that prepares data for Masked Language Modeling (MLM) — the task used to pre-train BERT.

The collator:

- takes tokenized examples,
- pads them,
- applies random masking,
- returns tensors ready for training.

```
data_collator = DataCollatorForLanguageModeling(
    tokenizer=bert_tokenizer,
    mlm_probability=0.15
)
```

This tells the collator:

- “Use this tokenizer”
- “Mask 15% of the tokens” (BERT default)

During masking, BERT uses the 80–10–10 rule:

For each selected token:

- 80% → replaced with [MASK]
- 10% → replaced with a random token
- 10% → left unchanged (the model still has to predict it)

This prevents the model from overfitting to the literal [MASK] token.

In [ ]:
```
print(bert_tokenizer.convert_ids_to_tokens(tokens['input_ids']))
print(bert_tokenizer.convert_ids_to_tokens(mlm_tokens['input_ids'][0]))
```

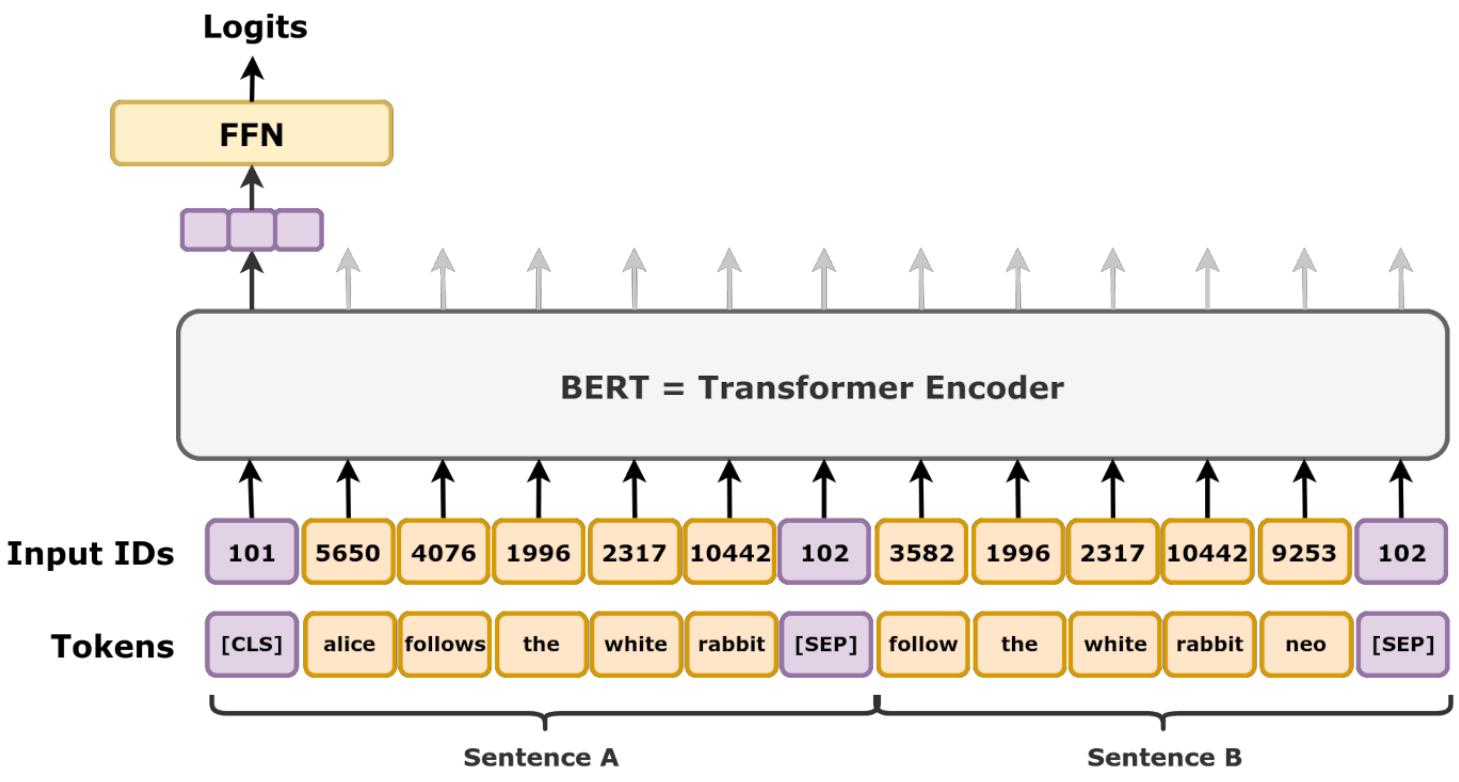
```
['[CLS]', 'alice', 'is', 'in', '##ex', '##pl', '##ica', '##bly', 'following', 'the', 'white', 'rabbit', '[SE
P]']
['[CLS]', 'alice', 'is', 'in', '##ex', '##pl', '[MASK]', '##bly', 'following', 'the', 'white', 'rabbit', '[SE
P]']
```

Next Sentence Prediction (NSP)

Another pre-training task is the Next Sentence Prediction (NSP) task. BERT was trained to predict if a second sentence is actually the next sentence in the original text or not.

In this way, the model learns the relationships between the sencences.

This task takes the special classifier token [CLS] (its final hidden states) as features for a classifier.



In [ ]:
```
bert_model.pooler
```

Out[ ]:
```
BertPooler(
  (dense): Linear(in_features=768, out_features=768, bias=True)
  (activation): Tanh()
)
```

```
In [ ]: sentence1 = 'alice follows the white rabbit'
        sentence2 = 'follow the white rabbit neo'
        bert_tokenizer(sentence1, sentence2, return_tensors='pt')
```

```
Out[ ]: {'input_ids': tensor([[  101,  5650,  4076,  1996,  2317, 10442,   102,  3582,  1996,  2317,
                 10442,  9253,   102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]]), 'attenti
        on_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

Outputs

```
In [ ]: sentence = 'And, so far as they knew, they were quite right' #train_dataset[100]['sentence']
        sentence
```

```
Out[ ]: 'And, so far as they knew, they were quite right'
```

```
In [ ]: tokens = bert_tokenizer(sentence,
                                padding='max_length',
                                max_length=30,
                                truncation=True,
                                return_tensors="pt")
        tokens
```

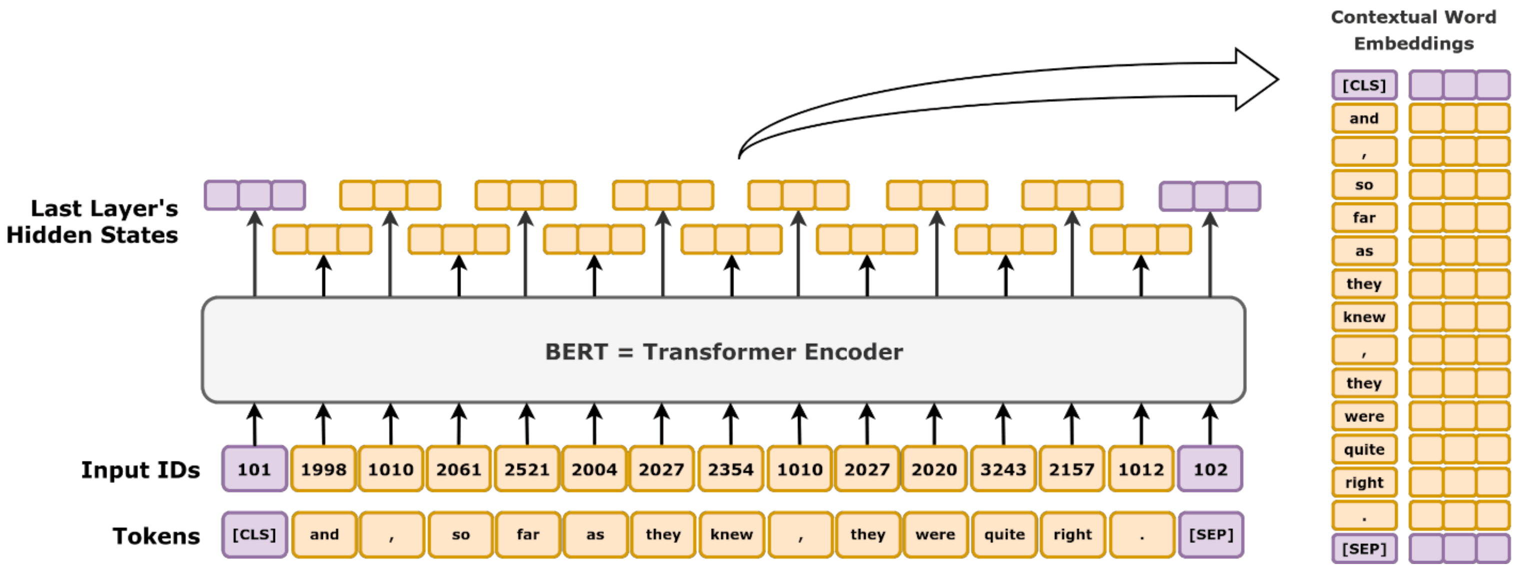
```
Out[ ]: {'input_ids': tensor([[ 101, 1998, 1010, 2061, 2521, 2004, 2027, 2354, 1010, 2027, 2020, 3243,
                 2157,  102,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                    0,    0,    0,    0,    0,    0]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0]])}
```

```
In [ ]: bert_model.eval()
        out = bert_model(input_ids=tokens['input_ids'],
                         attention_mask=tokens['attention_mask'],
                         output_attentions=True,
                         output_hidden_states=True,
                         return_dict=True)

        print()
        out.keys()
```

```
Out[ ]: odict_keys(['last_hidden_state', 'pooler_output', 'hidden_states', 'attentions'])
```

- `last_hidden_state` is returned by default and is the most importan output of the all: it contains the final hidden states for each and every token in the input, this can be seen as **contextual word embeddings**

- [CLS], [SEP], and [PAD] are also included



```
In [ ]: last_hidden_batch = out['last_hidden_state']
        last_hidden_sentence = last_hidden_batch[0]
        # Removes hidden states for [PAD] tokens using the mask
        mask = tokens['attention_mask'].squeeze().bool()
        embeddings = last_hidden_sentence[mask]
        # Removes embeddings for the first [CLS] and last [SEP] tokens
        embeddings[1:-1]
```

```
Out[ ]: tensor([[-0.0207,  0.7596, -0.3298,  ..., -0.0624,  0.5881, -0.3320],
                [-0.2141,  0.9708,  0.3524,  ..., -0.7776,  0.0939, -0.7059],
                [-0.1622,  0.5184,  0.2489,  ..., -0.6274, -0.3363,  0.3238],
                ...,
                [ 0.3006,  0.4791, -0.4585,  ..., -0.3678,  0.3518, -0.4639],
                [ 0.0714,  0.5561,  0.3115,  ..., -0.5760, -0.5615, -0.8796],
                [-0.3458,  0.0045,  0.2338,  ...,  0.5585,  0.5742, -0.8070]],
               grad_fn=<SliceBackward0>)
```

```
In [ ]: get_embeddings(bert_flair, sentence)
```

```
Out[ ]: tensor([[-0.0207,  0.7596, -0.3298,  ..., -0.0624,  0.5881, -0.3320],
                [-0.2141,  0.9708,  0.3524,  ..., -0.7776,  0.0939, -0.7059],
                [-0.1622,  0.5184,  0.2489,  ..., -0.6274, -0.3363,  0.3238],
                ...,
                [ 0.3006,  0.4791, -0.4585,  ..., -0.3678,  0.3518, -0.4639],
                [ 0.0714,  0.5561,  0.3115,  ..., -0.5760, -0.5615, -0.8796],
                [-0.3458,  0.0045,  0.2338,  ...,  0.5585,  0.5742, -0.8070]],
               device='cuda:0')
```

- `hidden_states` returns hidden states for every layer in BERT encoder architecture, including the last one, and the input embedding as well.

Therefore 12 +1 (the input embeddings)

```
In [ ]: print(len(out['hidden_states']))
        print(out['hidden_states'][0].shape)

        13
        torch.Size([1, 30, 768])
```

```
In [ ]: (out['hidden_states'][0] == bert_model.embeddings(tokens['input_ids'])).all()

Out[ ]: tensor(True)
```

```
In [ ]: (out['hidden_states'][-1] == out['last_hidden_state']).all()

Out[ ]: tensor(True)
```

- `pooler_output` is returned by default, it's the output of the pooler given the last hidden state as its input

```
In [ ]: (out['pooler_output'] == bert_model.pooler(out['last_hidden_state'])).all()

Out[ ]: tensor(True)
```

- `attentions` return the self-attention scores for each attention head in each layer of BERT's encoder:

```
In [ ]: print(len(out['attentions']))
        print(out['attentions'][0].shape)

        12
        torch.Size([1, 12, 30, 30])
```

```
In [ ]: print(type(out['attentions']))

        <class 'tuple'>
```

12 elements, one for each layer, each element has a tensor containing the scores for the sentences in the mini-batch (only one in our case). Those scores include each 12 self-attention heads, each head indicating how mcuh attention each of the 30 tokens is paying to all 30 tokens.

Model IV - Classifying using BERT

In [ ]:
```
class BERTClassifier(nn.Module):
    def __init__(self, bert_model, ff_units, n_outputs, dropout=0.3):
        super().__init__()
        self.d_model = bert_model.config.dim
        self.n_outputs = n_outputs
        self.encoder = bert_model
        self.mlp = nn.Sequential(
            nn.Linear(self.d_model, ff_units),
            nn.ReLU(),
            nn.Dropout(dropout),

            nn.Linear(ff_units, n_outputs)
        )

    def encode(self, source, source_mask=None):
        states = self.encoder(input_ids=source,
                              attention_mask=source_mask)[0]
        cls_state = states[:, 0]
        return cls_state

    def forward(self, X):
        source_mask = (X > 0)
        # Featurizer
        cls_state = self.encode(X, source_mask)
        # Classifier
        out = self.mlp(cls_state)
        return out
```

our model takes

- an instance of a pretrained BERT model.
- the desidered number of outputs (logits) corresponding to the number of classes
- the `forward()` takes mini-batch of token-ids, encodes them using BERT and outputs logits

Data Preparation

In [ ]:
```
def tokenize_dataset(hf_dataset, sentence_field, label_field, tokenizer, **kwargs):
    sentences = hf_dataset[sentence_field]
    token_ids = tokenizer(sentences, return_tensors='pt', **kwargs)['input_ids']
    labels = torch.as_tensor(hf_dataset[label_field])
    dataset = TensorDataset(token_ids, labels)
    return dataset
```

In [ ]:
```
auto_tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncased')
tokenizer_kwargs = dict(truncation=True, padding=True, max_length=30, add_special_tokens=True)
```

In [ ]:
```
train_dataset_float = train_dataset.map(lambda row: {'labels': [float(row['labels'])]})
test_dataset_float = test_dataset.map(lambda row: {'labels': [float(row['labels'])]})

train_tensor_dataset = tokenize_dataset(train_dataset_float, 'sentence', 'labels', auto_tokenizer, **tokenizer
_kwargs)
test_tensor_dataset = tokenize_dataset(test_dataset_float, 'sentence', 'labels', auto_tokenizer, **tokenizer_k
wargs)

generator = torch.Generator()
train_loader = DataLoader(train_tensor_dataset, batch_size=4, shuffle=True, generator=generator)
test_loader = DataLoader(test_tensor_dataset, batch_size=8)
```

Model Configuration & Training

In [ ]:
```
torch.manual_seed(41)
bert_model = AutoModel.from_pretrained("distilbert-base-uncased")
model = BERTClassifier(bert_model, 128, n_outputs=1)
loss_fn = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

In [ ]:
```
sbs_bert = StepByStep(model, loss_fn, optimizer)
sbs_bert.set_loaders(train_loader, test_loader)
sbs_bert.train(1)
```

In [ ]:
```
sbs_bert.count_parameters()
```

Out[ ]:
```
66461441
```

In [ ]:
```
StepByStep.loader_apply(test_loader, sbs_bert.correct)
```

Out[ ]:
```
tensor([[435, 444],
        [312, 327]])
```

Fine-Tuning with HuggingFace

As we said before, there is a BERT model for every task, and we need just to fine-tune it.

HF makes at our disposal a **trainer** to do most of the fine-tuning work.

- Pre-training tasks:
  - Masked language mode (`BertForMaskedLM`)
  - Next sentence prediction (`BertForNextSentencePrediction`)
- Typical tasks:
  - Sequence classification (`BertForSequenceClassification`)
  - Token classification (`BertForTokenClassification`)
  - Question answering (`BertForQuestionAnswering`)
- Others:
  - Multiple choice (`BertForMultipleChoice`)

  In our case, we want to use `DistilBERT` for sequence classification.

Sequence Classification (or Regression)

In [ ]:
```
from transformers import DistilBertForSequenceClassification
torch.manual_seed(42)
bert_cls = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=2)
```

In [ ]:
```
from transformers import AutoModelForSequenceClassification
auto_cls = AutoModelForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=2)
print(auto_cls.__class__)
```

```
<class 'transformers.models.distilbert.modeling_distilbert.DistilBertForSequenceClassification'>
```

What it does is to add a single linear layer (classifier) on top of the pooled output from the underlying model to produce the logits.

We have the model, we prepare the dataset...

We need to tokenize our HF's datasets, and we do it one row at the time creating a new column to contain the tokenized version of the sentence.

Tokenized Dataset

In [ ]:
```
auto_tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncased')
def tokenize(row):
    return auto_tokenizer(row['sentence'],
                          truncation=True,
                          padding='max_length',
                          max_length=30)
```

In [ ]:
```
tokenized_train_dataset = train_dataset.map(tokenize, batched=True)
tokenized_test_dataset = test_dataset.map(tokenize, batched=True)
```

```
In [ ]: print(tokenized_train_dataset[0])
```

```
{'sentence': 'Then he said:      "Do you suppose Oz could give me a heart?"', 'source': 'wizoz10-1740.txt', 'l
abels': 0, 'input_ids': [101, 2059, 2002, 2056, 1024, 1000, 2079, 2017, 6814, 11472, 2071, 2507, 2033, 1037, 2
540, 1029, 1000, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]}
```

```
In [ ]: '''
        we select only the need columns and return them as tensors
        '''

        tokenized_train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])
        tokenized_test_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])
```

```
In [ ]: tokenized_train_dataset[0]
```

```
Out[ ]: {'labels': tensor(0),
         'input_ids': tensor([  101,  2059,  2002,  2056,  1024,  1000,  2079,  2017,  6814, 11472,
                  2071,  2507,  2033,  1037,  2540,  1029,  1000,   102,     0,     0,
                     0,     0,     0,     0,     0,     0,     0,     0,     0,     0]),
         'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0])}
```

Trainer

```
In [ ]: from transformers import Trainer
        trainer = Trainer(model=bert_cls, train_dataset=tokenized_train_dataset)
```

```
/usr/local/lib/python3.12/dist-packages/accelerate/accelerator.py:446: FutureWarning: Passing the following ar
guments to `Accelerator` is deprecated and will be removed in version 1.0 of Accelerate: dict_keys(['dispatch_
batches', 'split_batches', 'even_batches', 'use_seedable_sampler']). Please pass an `accelerate.DataLoaderConf
iguration` instead:
dataloader_config = DataLoaderConfiguration(dispatch_batches=None, split_batches=False, even_batches=True, use
_seedable_sampler=True)
  warnings.warn(
```

In [ ]:
```
trainer.args
```

```
Out[ ]: TrainingArguments(
        _n_gpu=1,
        accelerator_config={'split_batches': False, 'dispatch_batches': None, 'even_batches': True, 'use_seedable_samp
        ler': True},
        adafactor=False,
        adam_beta1=0.9,
        adam_beta2=0.999,
        adam_epsilon=1e-08,
        auto_find_batch_size=False,
        bf16=False,
        bf16_full_eval=False,
        data_seed=None,
        dataloader_drop_last=False,
        dataloader_num_workers=0,
        dataloader_persistent_workers=False,
        dataloader_pin_memory=True,
        dataloader_prefetch_factor=None,
        ddp_backend=None,
        ddp_broadcast_buffers=None,
        ddp_bucket_cap_mb=None,
        ddp_find_unused_parameters=None,
        ddp_timeout=1800,
        debug=[],
        deepspeed=None,
        disable_tqdm=True,
        dispatch_batches=None,
        do_eval=False,
        do_predict=False,
        do_train=False,
        eval_accumulation_steps=None,
        eval_delay=0,
        eval_steps=None,
        evaluation_strategy=IntervalStrategy.NO,
        fp16=False,
        fp16_backend=auto,
        fp16_full_eval=False,
        fp16_opt_level=O1,
        fsdp=[],
        fsdp_config={'min_num_params': 0, 'xla': False, 'xla_fsdp_v2': False, 'xla_fsdp_grad_ckpt': False},
        fsdp_min_num_params=0,
        fsdp_transformer_layer_cls_to_wrap=None,
        full_determinism=False,
        gradient_accumulation_steps=1,
        gradient_checkpointing=False,
        gradient_checkpointing_kwargs=None,
        greater_is_better=None,
        group_by_length=False,
        half_precision_backend=auto,
        hub_always_push=False,
        hub_model_id=None,
        hub_private_repo=False,
        hub_strategy=HubStrategy.EVERY_SAVE,
        hub_token=<HUB_TOKEN>,
        ignore_data_skip=False,
        include_inputs_for_metrics=False,
        include_num_input_tokens_seen=False,
        include_tokens_per_second=False,
        jit_mode_eval=False,
        label_names=None,
        label_smoothing_factor=0.0,
        learning_rate=5e-05,
        length_column_name=length,
        load_best_model_at_end=False,
        local_rank=0,
        log_level=passive,
        log_level_replica=warning,
        log_on_each_node=True,
        logging_dir=tmp_trainer/runs/Nov25_10-56-45_053d52b2da13,
        logging_first_step=False,
        logging_nan_inf_filter=True,
        logging_steps=500,
        logging_strategy=IntervalStrategy.STEPS,
        lr_scheduler_kwargs={},
        lr_scheduler_type=SchedulerType.LINEAR,
        max_grad_norm=1.0,
        max_steps=-1,
        metric_for_best_model=None,
        mp_parameters=,
        neftune_noise_alpha=None,
        no_cuda=False,
        num_train_epochs=3.0,
        optim=OptimizerNames.ADAMW_TORCH,
        optim_args=None,
        output_dir=tmp_trainer,
        overwrite_output_dir=False,
        past_index=-1,
        per_device_eval_batch_size=8,
        per_device_train_batch_size=8,
        prediction_loss_only=False,
        push_to_hub=False,
        push_to_hub_model_id=None,
        push_to_hub_organization=None,
        push_to_hub_token=<PUSH_TO_HUB_TOKEN>,
        ray_scope=last,
        remove_unused_columns=True,
        report_to=['tensorboard', 'wandb'],
        resume_from_checkpoint=None,
        run_name=tmp_trainer,
        save_on_each_node=False,
        save_only_model=False,
        save_safetensors=True,
        save_steps=500,
        save_strategy=IntervalStrategy.STEPS,
        save_total_limit=None,
        seed=42,
        skip_memory_metrics=True,
        split_batches=None,
        tf32=None,
        torch_compile=False,
        torch_compile_backend=None,
        torch_compile_mode=None,
        torchdynamo=None,
        tpu_metrics_debug=False,
        tpu_num_cores=None,
        use_cpu=False,
        use_ipex=False,
        use_legacy_prediction_loop=False,
        use_mps_device=False,
        warmup_ratio=0.0,
        warmup_steps=0,
        weight_decay=0.0,
        )
```

In [ ]:
```python
from transformers import TrainingArguments
training_args = TrainingArguments(
    output_dir='./output', # Where to save the output files
    run_name="bert_experiment",  # Set a unique name for this run
    logging_dir="./logs",  # Directory for logging
    report_to=["none"],
    num_train_epochs=1,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=8,
    evaluation_strategy='steps',
    eval_steps=300,
    logging_steps=300,
    gradient_accumulation_steps=8,
)
```

Check the batch size, it is only one, but we keep accumulating the gradients for 8 steps, it is a way to simulate 8 size batches.

In [ ]:
```python
def compute_metrics(eval_pred):
    predictions = eval_pred.predictions
    labels = eval_pred.label_ids
    predictions = np.argmax(predictions, axis=1)
    return {"accuracy": (predictions == labels).mean()}
```

We can specify a class to compute the desired metrics and pass it to the Trainer instance

```
In [ ]: trainer = Trainer(model=bert_cls,
                          args=training_args,
                          train_dataset=tokenized_train_dataset,
                          eval_dataset=tokenized_test_dataset,
                          compute_metrics=compute_metrics)
```

```
/usr/local/lib/python3.12/dist-packages/accelerate/accelerator.py:446: FutureWarning: Passing the following ar
guments to `Accelerator` is deprecated and will be removed in version 1.0 of Accelerate: dict_keys(['dispatch_
batches', 'split_batches', 'even_batches', 'use_seedable_sampler']). Please pass an `accelerate.DataLoaderConf
iguration` instead:
dataloader_config = DataLoaderConfiguration(dispatch_batches=None, split_batches=False, even_batches=True, use
_seedable_sampler=True)
  warnings.warn(
```

```
In [ ]: trainer.train()
```

```
{'loss': 0.2006, 'grad_norm': 0.09020846337080002, 'learning_rate': 1.103896103896104e-05, 'epoch': 0.78}
{'eval_loss': 0.11388982087373734, 'eval_accuracy': 0.9623865110246433, 'eval_runtime': 0.9281, 'eval_samples_
per_second': 830.774, 'eval_steps_per_second': 104.52, 'epoch': 0.78}
{'train_runtime': 71.0087, 'train_samples_per_second': 43.389, 'train_steps_per_second': 5.422, 'train_loss':
0.17839183311957818, 'epoch': 1.0}
```

```
Out[ ]: TrainOutput(global_step=385, training_loss=0.17839183311957818, metrics={'train_runtime': 71.0087, 'train_samp
        les_per_second': 43.389, 'train_steps_per_second': 5.422, 'train_loss': 0.17839183311957818, 'epoch': 1.0})
```

```
In [ ]: trainer.evaluate()
```

```
{'eval_loss': 0.1219308078289032, 'eval_accuracy': 0.9662775616083009, 'eval_runtime': 0.7885, 'eval_samples_p
er_second': 977.864, 'eval_steps_per_second': 123.026, 'epoch': 1.0}
```

```
Out[ ]: {'eval_loss': 0.1219308078289032,
         'eval_accuracy': 0.9662775616083009,
         'eval_runtime': 0.7885,
         'eval_samples_per_second': 977.864,
         'eval_steps_per_second': 123.026,
         'epoch': 1.0}
```

We can save it, and use later.

```
In [ ]: trainer.save_model('bert_alice_vs_wizard')
        os.listdir('bert_alice_vs_wizard')
```

```
Out[ ]: ['training_args.bin', 'model.safetensors', 'config.json']
```

```
In [ ]: loaded_model = AutoModelForSequenceClassification.from_pretrained('bert_alice_vs_wizard')
        loaded_model.device
```

```
Out[ ]: device(type='cpu')
```

```
In [ ]: device = 'cuda' if torch.cuda.is_available() else 'cpu'
        loaded_model.to(device)
        loaded_model.device
```

```
Out[ ]: device(type='cuda', index=0)
```

Predictions

If you remember last time, we started with a peculiar sentence. Now we are able to classify it (also before, you can try).

We tokenize it, we send to the right device and then we evaluate

```
In [ ]: sentence = 'Down the yellow brick rabbit hole'
        tokens = auto_tokenizer(sentence, return_tensors='pt')
        tokens
```

```
Out[ ]: {'input_ids': tensor([[  101,  2091,  1996,  3756,  5318, 10442,  4920,   102]]), 'attention_mask': tensor
        ([[1, 1, 1, 1, 1, 1, 1, 1]])}
```

```
In [ ]: print(type(tokens))
        tokens.to(loaded_model.device)

        <class 'transformers.tokenization_utils_base.BatchEncoding'>

Out[ ]: {'input_ids': tensor([[  101,  2091,  1996,  3756,  5318, 10442,  4920,   102]],
               device='cuda:0'), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1]], device='cuda:0')}
```

```
In [ ]: loaded_model.eval()
        logits = loaded_model(input_ids=tokens['input_ids'], attention_mask=tokens['attention_mask'])
        logits

Out[ ]: SequenceClassifierOutput(loss=None, logits=tensor([[ 0.4209, -0.2158]], device='cuda:0', grad_fn=<AddmmBackwar
        d0>), hidden_states=None, attentions=None)
```

```
In [ ]: logits.logits.argmax(dim=1)

Out[ ]: tensor([0], device='cuda:0')
```

Pipeline

We can make it more efficient using pipelines

There are many pipelines, one for each task:

- `TextClassificationPipeline`
- `TextGenerationPipeline`

Every pipeline takes at least two argument:

- a model
- a tokenizer

Now, we can make predictions using the **original sentences**

```
In [ ]: from transformers import TextClassificationPipeline
        device_index = loaded_model.device.index if loaded_model.device.type != 'cpu' else -1
        classifier = TextClassificationPipeline(model=loaded_model,
                                                tokenizer=auto_tokenizer,
                                                device=device_index)
```

```
In [ ]: classifier(['Down the Yellow Brick Rabbit Hole', 'Alice rules!'])

Out[ ]: [{'label': 'LABEL_0', 'score': 0.6539878249168396},
         {'label': 'LABEL_1', 'score': 0.9958345890045166}]
```

```
In [ ]: loaded_model.config.id2label = {0: 'Wizard', 1: 'Alice'}
```

```
In [ ]: classifier(['Down the Yellow Brick Rabbit Hole', 'Alice rules!'])

Out[ ]: [{'label': 'Wizard', 'score': 0.6539878249168396},
         {'label': 'Alice', 'score': 0.9958345890045166}]
```

More Pipelines

It is possible to use pre-trained pipeline for **typical tasks** like sentiment analysis, without any fine-tuning.

*check the pipeline documentation on HuggingFace

```
In [ ]: from transformers import pipeline
        sentiment = pipeline('sentiment-analysis')

        /usr/local/lib/python3.12/dist-packages/huggingface_hub/file_download.py:942: FutureWarning: `resume_download`
        is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you want to forc
        e a new download, use `force_download=True`.
          warnings.warn(
In [ ]: sentence = train_dataset[0]['sentence']
        print(sentence)
        print(sentiment(sentence))
```

```
Then he said:      "Do you suppose Oz could give me a heart?"
[{'label': 'POSITIVE', 'score': 0.9901434183120728}]
```

```
In [ ]: from transformers.pipelines import SUPPORTED_TASKS
        # UPDATED
        ###########################################################
        # sentiment-analysis was replaced by text-classification
        # in the dictionary of supported tasks
        # SUPPORTED_TASKS['sentiment-analysis']
        SUPPORTED_TASKS['text-classification']
        ###########################################################
```

```
Out[ ]: {'impl': transformers.pipelines.text_classification.TextClassificationPipeline,
         'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForSequenceClassification,),
         'pt': (transformers.models.auto.modeling_auto.AutoModelForSequenceClassification,),
         'default': {'model': {'pt': ('distilbert/distilbert-base-uncased-finetuned-sst-2-english',
             'af0f99b'),
           'tf': ('distilbert/distilbert-base-uncased-finetuned-sst-2-english',
             'af0f99b')}},
         'type': 'text'}
```

In [ ]:
```
SUPPORTED_TASKS
```

```
Out[ ]: {'audio-classification': {'impl': transformers.pipelines.audio_classification.AudioClassificationPipeline,
          'tf': (),
          'pt': (transformers.models.auto.modeling_auto.AutoModelForAudioClassification,),
          'default': {'model': {'pt': ('superb/wav2vec2-base-superb-ks', '372e048')}},
          'type': 'audio'},
         'automatic-speech-recognition': {'impl': transformers.pipelines.automatic_speech_recognition.AutomaticSpeechR
        ecognitionPipeline,
          'tf': (),
          'pt': (transformers.models.auto.modeling_auto.AutoModelForCTC,
           transformers.models.auto.modeling_auto.AutoModelForSpeechSeq2Seq),
          'default': {'model': {'pt': ('facebook/wav2vec2-base-960h', '55bb623')}},
          'type': 'multimodal'},
         'text-to-audio': {'impl': transformers.pipelines.text_to_audio.TextToAudioPipeline,
          'tf': (),
          'pt': (transformers.models.auto.modeling_auto.AutoModelForTextToWaveform,
           transformers.models.auto.modeling_auto.AutoModelForTextToSpectrogram),
          'default': {'model': {'pt': ('suno/bark-small', '645cfba')}},
          'type': 'text'},
         'feature-extraction': {'impl': transformers.pipelines.feature_extraction.FeatureExtractionPipeline,
          'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModel,),
          'pt': (transformers.models.auto.modeling_auto.AutoModel,),
          'default': {'model': {'pt': ('distilbert/distilbert-base-cased', '935ac13'),
            'tf': ('distilbert/distilbert-base-cased', '935ac13')}},
          'type': 'multimodal'},
         'text-classification': {'impl': transformers.pipelines.text_classification.TextClassificationPipeline,
          'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForSequenceClassification,),
          'pt': (transformers.models.auto.modeling_auto.AutoModelForSequenceClassification,),
          'default': {'model': {'pt': ('distilbert/distilbert-base-uncased-finetuned-sst-2-english',
             'af0f99b'),
            'tf': ('distilbert/distilbert-base-uncased-finetuned-sst-2-english',
             'af0f99b')}},
          'type': 'text'},
         'token-classification': {'impl': transformers.pipelines.token_classification.TokenClassificationPipeline,
          'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForTokenClassification,),
          'pt': (transformers.models.auto.modeling_auto.AutoModelForTokenClassification,),
          'default': {'model': {'pt': ('dbmdz/bert-large-cased-finetuned-conll03-english',
             'f2482bf'),
            'tf': ('dbmdz/bert-large-cased-finetuned-conll03-english', 'f2482bf')}},
          'type': 'text'},
         'question-answering': {'impl': transformers.pipelines.question_answering.QuestionAnsweringPipeline,
          'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForQuestionAnswering,),
          'pt': (transformers.models.auto.modeling_auto.AutoModelForQuestionAnswering,),
          'default': {'model': {'pt': ('distilbert/distilbert-base-cased-distilled-squad',
             '626af31'),
            'tf': ('distilbert/distilbert-base-cased-distilled-squad', '626af31')}},
          'type': 'text'},
         'table-question-answering': {'impl': transformers.pipelines.table_question_answering.TableQuestionAnsweringPi
        peline,
          'pt': (transformers.models.auto.modeling_auto.AutoModelForTableQuestionAnswering,),
          'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForTableQuestionAnswering,),
          'default': {'model': {'pt': ('google/tapas-base-finetuned-wtq', '69ceee2'),
            'tf': ('google/tapas-base-finetuned-wtq', '69ceee2')}},
          'type': 'text'},
         'visual-question-answering': {'impl': transformers.pipelines.visual_question_answering.VisualQuestionAnswerin
        gPipeline,
          'pt': (transformers.models.auto.modeling_auto.AutoModelForVisualQuestionAnswering,),
          'tf': (),
          'default': {'model': {'pt': ('dandelin/vilt-b32-finetuned-vqa', '4355f59')}},
          'type': 'multimodal'},
         'document-question-answering': {'impl': transformers.pipelines.document_question_answering.DocumentQuestionAn
        sweringPipeline,
          'pt': (transformers.models.auto.modeling_auto.AutoModelForDocumentQuestionAnswering,),
          'tf': (),
          'default': {'model': {'pt': ('impira/layoutlm-document-qa', '52e01b3')}},
          'type': 'multimodal'},
         'fill-mask': {'impl': transformers.pipelines.fill_mask.FillMaskPipeline,
          'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForMaskedLM,),
          'pt': (transformers.models.auto.modeling_auto.AutoModelForMaskedLM,),
          'default': {'model': {'pt': ('distilbert/distilroberta-base', 'ec58a5b'),
            'tf': ('distilbert/distilroberta-base', 'ec58a5b')}},
          'type': 'text'},
         'summarization': {'impl': transformers.pipelines.text2text_generation.SummarizationPipeline,
          'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForSeq2SeqLM,),
          'pt': (transformers.models.auto.modeling_auto.AutoModelForSeq2SeqLM,),
          'default': {'model': {'pt': ('sshleifer/distilbart-cnn-12-6', 'a4f8f3e'),
            'tf': ('google-t5/t5-small', 'd769bba')}},
          'type': 'text'},
         'translation': {'impl': transformers.pipelines.text2text_generation.TranslationPipeline,
          'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForSeq2SeqLM,),
          'pt': (transformers.models.auto.modeling_auto.AutoModelForSeq2SeqLM,),
          'default': {('en',
            'fr'): {'model': {'pt': ('google-t5/t5-base', '686f1db'),
      'tf': ('google-t5/t5-base', '686f1db')}},
    ('en',
     'de'): {'model': {'pt': ('google-t5/t5-base', '686f1db'),
      'tf': ('google-t5/t5-base', '686f1db')}},
    ('en',
     'ro'): {'model': {'pt': ('google-t5/t5-base', '686f1db'),
      'tf': ('google-t5/t5-base', '686f1db')}}},
   'type': 'text'},
  'text2text-generation': {'impl': transformers.pipelines.text2text_generation.Text2TextGenerationPipeline,
   'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForSeq2SeqLM,),
   'pt': (transformers.models.auto.modeling_auto.AutoModelForSeq2SeqLM,),
   'default': {'model': {'pt': ('google-t5/t5-base', '686f1db'),
     'tf': ('google-t5/t5-base', '686f1db')}},
   'type': 'text'},
  'text-generation': {'impl': transformers.pipelines.text_generation.TextGenerationPipeline,
   'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForCausalLM,),
   'pt': (transformers.models.auto.modeling_auto.AutoModelForCausalLM,),
   'default': {'model': {'pt': ('openai-community/gpt2', '6c0e608'),
     'tf': ('openai-community/gpt2', '6c0e608')}},
   'type': 'text'},
  'zero-shot-classification': {'impl': transformers.pipelines.zero_shot_classification.ZeroShotClassificationPi
peline,
   'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForSequenceClassification,),
   'pt': (transformers.models.auto.modeling_auto.AutoModelForSequenceClassification,),
   'default': {'model': {'pt': ('facebook/bart-large-mnli', 'c626438'),
     'tf': ('FacebookAI/roberta-large-mnli', '130fb28')},
    'config': {'pt': ('facebook/bart-large-mnli', 'c626438'),
     'tf': ('FacebookAI/roberta-large-mnli', '130fb28')}},
   'type': 'text'},
  'zero-shot-image-classification': {'impl': transformers.pipelines.zero_shot_image_classification.ZeroShotImag
eClassificationPipeline,
   'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForZeroShotImageClassification,),
   'pt': (transformers.models.auto.modeling_auto.AutoModelForZeroShotImageClassification,),
   'default': {'model': {'pt': ('openai/clip-vit-base-patch32', 'f4881ba'),
     'tf': ('openai/clip-vit-base-patch32', 'f4881ba')}},
   'type': 'multimodal'},
  'zero-shot-audio-classification': {'impl': transformers.pipelines.zero_shot_audio_classification.ZeroShotAudi
oClassificationPipeline,
   'tf': (),
   'pt': (transformers.models.auto.modeling_auto.AutoModel,),
   'default': {'model': {'pt': ('laion/clap-htsat-fused', '973b6e5')}},
   'type': 'multimodal'},
  'conversational': {'impl': transformers.pipelines.conversational.ConversationalPipeline,
   'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForSeq2SeqLM,
    transformers.models.auto.modeling_tf_auto.TFAutoModelForCausalLM),
   'pt': (transformers.models.auto.modeling_auto.AutoModelForSeq2SeqLM,
    transformers.models.auto.modeling_auto.AutoModelForCausalLM),
   'default': {'model': {'pt': ('microsoft/DialoGPT-medium', '8bada3b'),
     'tf': ('microsoft/DialoGPT-medium', '8bada3b')}},
   'type': 'text'},
  'image-classification': {'impl': transformers.pipelines.image_classification.ImageClassificationPipeline,
   'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForImageClassification,),
   'pt': (transformers.models.auto.modeling_auto.AutoModelForImageClassification,),
   'default': {'model': {'pt': ('google/vit-base-patch16-224', '5dca96d'),
     'tf': ('google/vit-base-patch16-224', '5dca96d')}},
   'type': 'image'},
  'image-feature-extraction': {'impl': transformers.pipelines.image_feature_extraction.ImageFeatureExtractionPi
peline,
   'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModel,),
   'pt': (transformers.models.auto.modeling_auto.AutoModel,),
   'default': {'model': {'pt': ('google/vit-base-patch16-224', '29e7a1e183'),
     'tf': ('google/vit-base-patch16-224', '29e7a1e183')}},
   'type': 'image'},
  'image-segmentation': {'impl': transformers.pipelines.image_segmentation.ImageSegmentationPipeline,
   'tf': (),
   'pt': (transformers.models.auto.modeling_auto.AutoModelForImageSegmentation,
    transformers.models.auto.modeling_auto.AutoModelForSemanticSegmentation),
   'default': {'model': {'pt': ('facebook/detr-resnet-50-panoptic',
      'fc15262')}},
   'type': 'multimodal'},
  'image-to-text': {'impl': transformers.pipelines.image_to_text.ImageToTextPipeline,
   'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForVision2Seq,),
   'pt': (transformers.models.auto.modeling_auto.AutoModelForVision2Seq,),
   'default': {'model': {'pt': ('ydshieh/vit-gpt2-coco-en', '65636df'),
     'tf': ('ydshieh/vit-gpt2-coco-en', '65636df')}},
   'type': 'multimodal'},
  'object-detection': {'impl': transformers.pipelines.object_detection.ObjectDetectionPipeline,
   'tf': (),
   'pt': (transformers.models.auto.modeling_auto.AutoModelForObjectDetection,),
   'default': {'model': {'pt': ('facebook/detr-resnet-50', '2729413')}},
   'type': 'multimodal'},
  'zero-shot-object-detection': {'impl': transformers.pipelines.zero_shot_object_detection.ZeroShotObjectDetect
```

```
ionPipeline,
  'tf': (),
  'pt': (transformers.models.auto.modeling_auto.AutoModelForZeroShotObjectDetection,),
  'default': {'model': {'pt': ('google/owlvit-base-patch32', '17740e1')}},
  'type': 'multimodal'},
 'depth-estimation': {'impl': transformers.pipelines.depth_estimation.DepthEstimationPipeline,
  'tf': (),
  'pt': (transformers.models.auto.modeling_auto.AutoModelForDepthEstimation,),
  'default': {'model': {'pt': ('Intel/dpt-large', 'e93beec')}},
  'type': 'image'},
 'video-classification': {'impl': transformers.pipelines.video_classification.VideoClassificationPipeline,
  'tf': (),
  'pt': (transformers.models.auto.modeling_auto.AutoModelForVideoClassification,),
  'default': {'model': {'pt': ('MCG-NJU/videomae-base-finetuned-kinetics',
     '4800870')}},
  'type': 'video'},
 'mask-generation': {'impl': transformers.pipelines.mask_generation.MaskGenerationPipeline,
  'tf': (),
  'pt': (transformers.models.auto.modeling_auto.AutoModelForMaskGeneration,),
  'default': {'model': {'pt': ('facebook/sam-vit-huge', '997b15')}},
  'type': 'multimodal'},
 'image-to-image': {'impl': transformers.pipelines.image_to_image.ImageToImagePipeline,
  'tf': (),
  'pt': (transformers.models.auto.modeling_auto.AutoModelForImageToImage,),
  'default': {'model': {'pt': ('caidas/swin2SR-classical-sr-x2-64',
     '4aaedcb')}},
  'type': 'image'}}
```

```
In [ ]: SUPPORTED_TASKS['text-generation']
```

```
Out[ ]: {'impl': transformers.pipelines.text_generation.TextGenerationPipeline,
         'tf': (transformers.models.auto.modeling_tf_auto.TFAutoModelForCausalLM,),
         'pt': (transformers.models.auto.modeling_auto.AutoModelForCausalLM,),
         'default': {'model': {'pt': ('openai-community/gpt2', '6c0e608'),
           'tf': ('openai-community/gpt2', '6c0e608')}},
         'type': 'text'}
```

GPT-2

The **G**enerative **P**retrained **T**ransfomer 2 is able to generate text.

It was trained to fill the in the blanks at the end of the sentences, effectively predicting the next word in a given sentence.

This taks is exactly what a transformer Decoder does, and this what GPT-2 is, a transfomer decoder.

40GB of internet text, 8 millions of web pages, 48 layers, 12 attention heads, and 1600 hidden dimensions, 1.5 billion parameters (Nov. 2019).

```
In [ ]: text_generator = pipeline("text-generation")
```

```
In [ ]: text_generator.model.config.task_specific_params
```

```
Out[ ]: {'text-generation': {'do_sample': True, 'max_length': 50}}
```

```
In [ ]: base_text = """
        Alice was beginning to get very tired of sitting by her sister on the bank,
        and of having nothing to do:  once or twice she had peeped into the book her
         sister was reading, but it had no pictures or conversations in it, `and what
         is the use of a book,'thought Alice `without pictures or conversation?'
         So she was considering in her own mind (as well as she could, for the hot day
         made her feel very sleepy and stupid), whether the pleasure of making a
         daisy-chain would be worth the trouble of getting up and picking the daisies,
         when suddenly a White Rabbit with pink eyes ran close by her.
        """
```

```
In [ ]: result = text_generator(base_text, max_length=250)
        print(result[0]['generated_text'])
```

```
Alice was beginning to get very tired of sitting by her sister on the bank,
and of having nothing to do:  once or twice she had peeped into the book her
 sister was reading, but it had no pictures or conversations in it, `and what
 is the use of a book,'thought Alice `without pictures or conversation?'
 So she was considering in her own mind (as well as she could, for the hot day
 made her feel very sleepy and stupid), whether the pleasure of making a
 daisy-chain would be worth the trouble of getting up and picking the daisies,
 when suddenly a White Rabbit with pink eyes ran close by her.
`My friend, why don't you come back here, there's no danger waiting there so

many hours?' asked Alice. `You look a bit tired, when you're all about the same time as you were before.

'I'm afraid I can't look myself in the mirror at midnight, I think, and it's better to wear a coat on.

'Now, I don't want to think about it. You should be thinking about it.'

By this time there were still some small
```

Hold-on, we can fine-tune GPT2 too.

Data Preparation

```
In [ ]: dataset = load_dataset(path='csv', data_files=['texts/alice28-1476.sent.csv'], quotechar='\\', split=Split.TRA
        IN)
```

```
In [ ]: shuffled_dataset = dataset.shuffle(seed=42)
        split_dataset = shuffled_dataset.train_test_split(test_size=0.2, seed=42)
        train_dataset, test_dataset = split_dataset['train'], split_dataset['test']
```

```
In [ ]: auto_tokenizer = AutoTokenizer.from_pretrained('gpt2')
        def tokenize(row):
            return auto_tokenizer(row['sentence'])
```

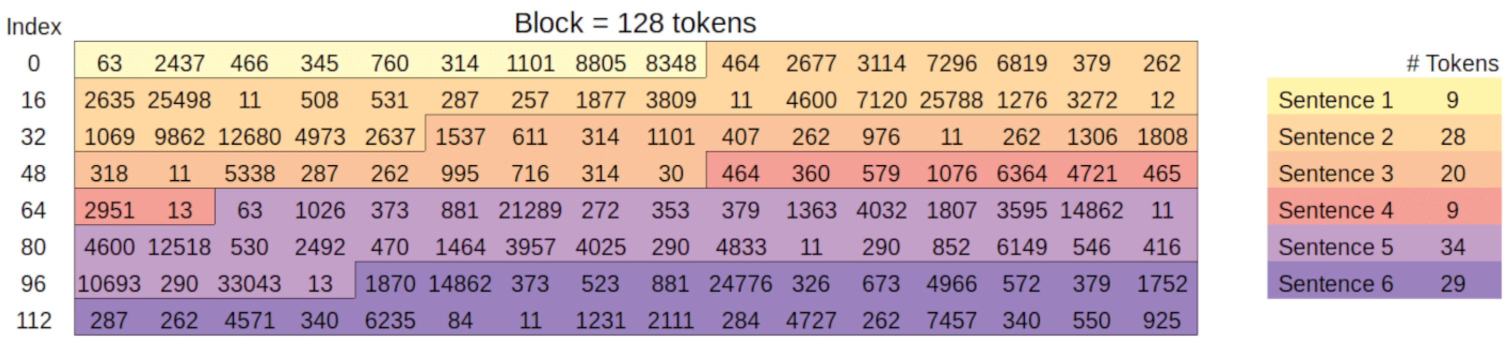
```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/file_download.py:942: FutureWarning: `resume_download`
is deprecated and will be removed in version 1.0.0. Downloads always resume when possible. If you want to forc
e a new download, use `force_download=True`.
  warnings.warn(
```

- GPT2 uses a different pre-trained tokenizer based on Byte-Pair encoding
- we don't need padding, we need to generate text, and we don't want to write something after many padding tokens.
- we remove (below) the source and sentence columns (as before)
- then we pack sentence together, concateneting the inputs and chunk them into blocks.

```
In [ ]: tokenized_train_dataset = train_dataset.map(tokenize, remove_columns=['source', 'sentence'], batched=True)
        tokenized_test_dataset = test_dataset.map(tokenize, remove_columns=['source', 'sentence'], batched=True)
```

```
In [ ]: list(map(len, tokenized_train_dataset[0:6]['input_ids']))

Out[ ]: [9, 28, 20, 9, 34, 29]
```

"Packed" Dataset


Block = 128 tokens

Index																
0	63	2437	466	345	760	314	1101	8805	8348	464	2677	3114	7296	6819	379	262
16	2635	25498	11	508	531	287	257	1877	3809	11	4600	7120	25788	1276	3272	12
32	1069	9862	12680	4973	2637	1537	611	314	1101	407	262	976	11	262	1306	1808
48	318	11	5338	287	262	995	716	314	30	464	360	579	1076	6364	4721	465
64	2951	13	63	1026	373	881	21289	272	353	379	1363	4032	1807	3595	14862	11
80	4600	12518	530	2492	470	1464	3957	4025	290	4833	11	290	852	6149	546	416
96	10693	290	33043	13	1870	14862	373	523	881	24776	326	673	4966	572	379	1752
112	287	262	4571	340	6235	84	11	1231	2111	284	4727	262	7457	340	550	925

	# Tokens
Sentence 1	9
Sentence 2	28
Sentence 3	20
Sentence 4	9
Sentence 5	34
Sentence 6	29

```
In [ ]: # Adapted from https://github.com/huggingface/transformers/blob/master/examples/pytorch/language-modeling/run_
        clm.py
        def group_texts(examples, block_size=128):
            # Concatenate all texts.
            concatenated_examples = {k: sum(examples[k], []) for k in examples.keys()}
            total_length = len(concatenated_examples[list(examples.keys())[0]])
            # We drop the small remainder, we could add padding if the model supported it instead of this drop, you ca
        n
            # customize this part to your needs.
            total_length = (total_length // block_size) * block_size
            # Split by chunks of max_len.
            result = {
                k: [t[i : i + block_size] for i in range(0, total_length, block_size)]
                for k, t in concatenated_examples.items()
            }
            result["labels"] = result["input_ids"].copy()
            return result
```

```
In [ ]: lm_train_dataset = tokenized_train_dataset.map(group_texts, batched=True)
        lm_test_dataset = tokenized_test_dataset.map(group_texts, batched=True)
        lm_train_dataset.set_format(type='torch')
        lm_test_dataset.set_format(type='torch')
```

```
In [ ]: print(lm_train_dataset[0]['input_ids'])

        tensor([   63,  2437,   466,   345,   760,   314,  1101,  8805,  8348,   464,
                 2677,  3114,  7296,  6819,   379,   262,  2635, 25498,    11,   508,
                  531,   287,   257,  1877,  3809,    11,  4600,  7120, 25788,  1276,
                 3272,    12,  1069,  9862, 12680,  4973,  2637,  1537,   611,   314,
                 1101,   407,   262,   976,    11,   262,  1306,  1808,   318,    11,
                 5338,   287,   262,   995,   716,   314,    30,   464,   360,   579,
                 1076,  6364,  4721,   465,  2951,    13,    63,  1026,   373,   881,
                21289,   272,   353,   379,  1363,  4032,  1807,  3595, 14862,    11,
                 4600, 12518,   530,  2492,   470,  1464,  3957,  4025,   290,  4833,
                   11,   290,   852,  6149,   546,   416, 10693,   290, 33043,    13,
                 1870, 14862,   373,   523,   881, 24776,   326,   673,  4966,   572,
                  379,  1752,   287,   262,  4571,   340,  6235,   284,    11,  1231,
                 2111,   284,  4727,   262,  7457,   340,   550,   925])
```

```
In [ ]: len(lm_train_dataset), len(lm_test_dataset)

Out[ ]: (239, 56)
```

Model Configuration & Training

GPT2 is a causal language modeling, therefore we use it to import

```
In [ ]: from transformers import AutoModelForCausalLM
        model = AutoModelForCausalLM.from_pretrained('gpt2')
        print(model.__class__)

        <class 'transformers.models.gpt2.modeling_gpt2.GPT2LMHeadModel'>
```

```
In [ ]: model.resize_token_embeddings(len(auto_tokenizer))
```

```
Out[ ]: Embedding(50257, 768)
```

```
In [ ]: training_args = TrainingArguments(
            output_dir='./output', # Where to save the output files
            run_name="gpt2_experiment",  # Set a unique name for this run
            logging_dir="./logs",  # Directory for logging
            report_to=["none"],
            num_train_epochs=1,
            per_device_train_batch_size=1,
            per_device_eval_batch_size=8,
            evaluation_strategy='steps',
            eval_steps=50,
            logging_steps=50,
            gradient_accumulation_steps=4,
            prediction_loss_only=True,
        )

        trainer = Trainer(model=model,
                          args=training_args,
                          train_dataset=lm_train_dataset,
                          eval_dataset=lm_test_dataset)
```

```
/usr/local/lib/python3.12/dist-packages/accelerate/accelerator.py:446: FutureWarning: Passing the following ar
guments to `Accelerator` is deprecated and will be removed in version 1.0 of Accelerate: dict_keys(['dispatch_
batches', 'split_batches', 'even_batches', 'use_seedable_sampler']). Please pass an `accelerate.DataLoaderConf
iguration` instead:
dataloader_config = DataLoaderConfiguration(dispatch_batches=None, split_batches=False, even_batches=True, use
_seedable_sampler=True)
  warnings.warn(
```

```
In [ ]: trainer.train()
```

```
{'loss': 3.567, 'grad_norm': 11.076814651489258, 'learning_rate': 7.627118644067798e-06, 'epoch': 0.84}
{'eval_loss': 3.336221933364868, 'eval_runtime': 0.5841, 'eval_samples_per_second': 95.873, 'eval_steps_per_se
cond': 11.984, 'epoch': 0.84}
{'train_runtime': 14.5277, 'train_samples_per_second': 16.451, 'train_steps_per_second': 4.061, 'train_loss':
3.5485167745816506, 'epoch': 0.99}
```

```
Out[ ]: TrainOutput(global_step=59, training_loss=3.5485167745816506, metrics={'train_runtime': 14.5277, 'train_sample
s_per_second': 16.451, 'train_steps_per_second': 4.061, 'train_loss': 3.5485167745816506, 'epoch': 0.99})
```

```
In [ ]: trainer.evaluate()
```

```
{'eval_loss': 3.329355478286743, 'eval_runtime': 0.582, 'eval_samples_per_second': 96.217, 'eval_steps_per_sec
ond': 12.027, 'epoch': 0.99}
```

```
Out[ ]: {'eval_loss': 3.329355478286743,
         'eval_runtime': 0.582,
         'eval_samples_per_second': 96.217,
         'eval_steps_per_second': 12.027,
         'epoch': 0.99}
```

Generating Text

```
In [ ]: device_index = model.device.index if model.device.type != 'cpu' else -1
        gpt2_gen = pipeline('text-generation', model=model, tokenizer=auto_tokenizer, device=device_index)
```

```
In [ ]: result = gpt2_gen(base_text, max_length=250)
        print(result[0]['generated_text'])
```

```
Alice was beginning to get very tired of sitting by her sister on the bank,
and of having nothing to do:  once or twice she had peeped into the book her
 sister was reading, but it had no pictures or conversations in it, `and what
 is the use of a book,'thought Alice `without pictures or conversation?'
 So she was considering in her own mind (as well as she could, for the hot day
 made her feel very sleepy and stupid), whether the pleasure of making a
 daisy-chain would be worth the trouble of getting up and picking the daisies,
 when suddenly a White Rabbit with pink eyes ran close by her.
'You never liked the sight,'said Alice.I shall never be so much frightened as you,' the
Alice said in a low voice.Alice followed the White Rabbit out by the street, as the great wall on the opposite
side of
`was a great white rabbit--' The Rabbit was not there; and the little fish made a few squeaks. Alice did not w
ant to be near it
```

```
In [ ]:
```