A2_part2_clinical_Word2Vec_embeddings_and_readmission_prediction

November 2, 2022

1 Assignment 2 - part 2 - Clincial Word Embeddings For Prediction

In part 1 you used structured sequence data to make predictions. In part 2 we will ignore that structured data and only use unstructured clinician notes from MIMIC-III. We will use discharge summaries to predict 30-day hospital readmission.

Importantly there are two separate distinct steps: 1. Learn good word embeddings. Word embeddings are function that maps words to fixed-length vectors (e.g. 32-dims). We want words that are similar in meaning to similar vector embeddings. 2. Create a deep learning model that takes text input and predicts whether a patient is readmitted. The inputs to the model will be word embeddings from the first step.

We will approach task 1, learning word embeddings, using the popular Word2Vec algorithm (see original paper). We'll use the skip-gram version of Word2Vec (the other version is 'continuous bag of words')

We will approach task 2 with an LSTM.

Q2.1

Explain how a model for 30-day readmission prediction could be used by doctors in a clinical setting.

1.0.1 Written answer: Doctors could use a 30-day readmission prediction model to assess patient care, devise prevention strategy, and improve resource allocation. Doctors could identify patients which would benefit most from care transition interventions, including increased at home surveillance. Additionally, this data could help improve patient care, as doctors could recognize less effective treatment strategies to minimize readmission. Lowering readmissions will also lower healthcare costs by significant margins.

Install the following packages.

[1]:
```
!pip install gensim
!pip install spacy==2.3.7
!pip install scispacy==0.3.0
!pip install nltk
!pip install tdqm
```

```
!pip install matplotlib
!pip install https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.2.5/
↪en_core_sci_md-0.2.5.tar.gz
!python -m spacy download en_core_web_sm
```

```
Requirement already satisfied: gensim in /opt/conda/lib/python3.7/site-packages
(4.2.0)
Requirement already satisfied: scipy>=0.18.1 in /opt/conda/lib/python3.7/site-
packages (from gensim) (1.7.3)
Requirement already satisfied: numpy>=1.17.0 in /opt/conda/lib/python3.7/site-
packages (from gensim) (1.19.5)
Requirement already satisfied: smart-open>=1.8.1 in
/opt/conda/lib/python3.7/site-packages (from gensim) (6.2.0)
Requirement already satisfied: spacy==2.3.7 in /opt/conda/lib/python3.7/site-
packages (2.3.7)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (0.7.9)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (2.0.7)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (1.1.3)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (0.10.1)
Requirement already satisfied: thinc<7.5.0,>=7.4.1 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (7.4.6)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (2.28.1)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (1.0.6)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.7/site-
packages (from spacy==2.3.7) (59.8.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (4.64.1)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (1.0.9)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (3.0.8)
Requirement already satisfied: numpy>=1.15.0 in /opt/conda/lib/python3.7/site-
packages (from spacy==2.3.7) (1.19.5)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (1.0.2)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from catalogue<1.1.0,>=0.0.7->spacy==2.3.7) (3.10.0)
Requirement already satisfied: typing-extensions>=3.6.4 in
/opt/conda/lib/python3.7/site-packages (from
catalogue<1.1.0,>=0.0.7->spacy==2.3.7) (3.10.0.2)
Requirement already satisfied: charset-normalizer<3,>=2 in
```

```
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0,>=2.13.0->spacy==2.3.7) (2.1.1)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0,>=2.13.0->spacy==2.3.7) (2022.9.24)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.7/site-
packages (from requests<3.0.0,>=2.13.0->spacy==2.3.7) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0,>=2.13.0->spacy==2.3.7) (1.26.11)
Requirement already satisfied: scispacy==0.3.0 in /opt/conda/lib/python3.7/site-
packages (0.3.0)
Requirement already satisfied: spacy<3.0.0,>=2.3.0 in
/opt/conda/lib/python3.7/site-packages (from scispacy==0.3.0) (2.3.7)
Requirement already satisfied: joblib in /opt/conda/lib/python3.7/site-packages
(from scispacy==0.3.0) (1.2.0)
Requirement already satisfied: pysbd in /opt/conda/lib/python3.7/site-packages
(from scispacy==0.3.0) (0.3.4)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages
(from scispacy==0.3.0) (1.19.5)
Requirement already satisfied: scikit-learn>=0.20.3 in
/opt/conda/lib/python3.7/site-packages (from scispacy==0.3.0) (1.0.2)
Requirement already satisfied: nmslib>=1.7.3.6 in /opt/conda/lib/python3.7/site-
packages (from scispacy==0.3.0) (2.1.1)
Requirement already satisfied: requests<3.0.0conllu,>=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from scispacy==0.3.0) (2.28.1)
Requirement already satisfied: psutil in /opt/conda/lib/python3.7/site-packages
(from nmslib>=1.7.3.6->scispacy==0.3.0) (5.9.3)
Requirement already satisfied: pybind11<2.6.2 in /opt/conda/lib/python3.7/site-
packages (from nmslib>=1.7.3.6->scispacy==0.3.0) (2.6.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.7/site-
packages (from requests<3.0.0conllu,>=2.0.0->scispacy==0.3.0) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0conllu,>=2.0.0->scispacy==0.3.0) (2022.9.24)
Requirement already satisfied: charset-normalizer<3,>=2 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0conllu,>=2.0.0->scispacy==0.3.0) (2.1.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0conllu,>=2.0.0->scispacy==0.3.0) (1.26.11)
Requirement already satisfied: scipy>=1.1.0 in /opt/conda/lib/python3.7/site-
packages (from scikit-learn>=0.20.3->scispacy==0.3.0) (1.7.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from scikit-
learn>=0.20.3->scispacy==0.3.0) (3.1.0)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/opt/conda/lib/python3.7/site-packages (from
```

```
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (2.0.7)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in
/opt/conda/lib/python3.7/site-packages (from
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (1.0.6)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/opt/conda/lib/python3.7/site-packages (from
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (3.0.8)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (0.7.9)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/opt/conda/lib/python3.7/site-packages (from
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (4.64.1)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in
/opt/conda/lib/python3.7/site-packages (from
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (1.1.3)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/opt/conda/lib/python3.7/site-packages (from
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (1.0.9)
Requirement already satisfied: thinc<7.5.0,>=7.4.1 in
/opt/conda/lib/python3.7/site-packages (from
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (7.4.6)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (0.10.1)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in
/opt/conda/lib/python3.7/site-packages (from
spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (1.0.2)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.7/site-
packages (from spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (59.8.0)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from catalogue<1.1.0,>=0.0.7->spacy<3.0.0,>=2.3.0->scispacy==0.3.0)
(3.10.0)
Requirement already satisfied: typing-extensions>=3.6.4 in
/opt/conda/lib/python3.7/site-packages (from
catalogue<1.1.0,>=0.0.7->spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (3.10.0.2)
Requirement already satisfied: nltk in /opt/conda/lib/python3.7/site-packages
(3.7)
Requirement already satisfied: joblib in /opt/conda/lib/python3.7/site-packages
(from nltk) (1.2.0)
Requirement already satisfied: click in /opt/conda/lib/python3.7/site-packages
(from nltk) (8.1.3)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages
(from nltk) (4.64.1)
Requirement already satisfied: regex>=2021.8.3 in /opt/conda/lib/python3.7/site-
packages (from nltk) (2022.9.13)
Requirement already satisfied: importlib-metadata in
/opt/conda/lib/python3.7/site-packages (from click->nltk) (4.11.4)
```

```
Requirement already satisfied: typing-extensions>=3.6.4 in
/opt/conda/lib/python3.7/site-packages (from importlib-metadata->click->nltk)
(3.10.0.2)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from importlib-metadata->click->nltk) (3.10.0)
Requirement already satisfied: tdqm in /opt/conda/lib/python3.7/site-packages
(0.0.1)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages
(from tdqm) (4.64.1)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.7/site-
packages (3.5.3)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.7/site-
packages (from matplotlib) (21.3)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-
packages (from matplotlib) (9.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.7/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.7/site-
packages (from matplotlib) (1.19.5)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-
packages (from matplotlib) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.7/site-packages (from matplotlib) (4.38.0)
Requirement already satisfied: typing-extensions in
/opt/conda/lib/python3.7/site-packages (from kiwisolver>=1.0.1->matplotlib)
(3.10.0.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-
packages (from python-dateutil>=2.7->matplotlib) (1.15.0)
Collecting https://s3-us-
west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.2.5/en_core_sci_md-0.2.5.tar.gz
  Using cached https://s3-us-
west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.2.5/en_core_sci_md-0.2.5.tar.gz
(79.9 MB)
  Preparing metadata (setup.py) … done
Requirement already satisfied: spacy>=2.3.0 in
/opt/conda/lib/python3.7/site-packages (from en-core-sci-md==0.2.5) (2.3.7)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (0.7.9)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (0.10.1)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
```

```
md==0.2.5) (4.64.1)
Requirement already satisfied: thinc<7.5.0,>=7.4.1 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (7.4.6)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (1.1.3)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.7/site-
packages (from spacy>=2.3.0->en-core-sci-md==0.2.5) (59.8.0)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (2.0.7)
Requirement already satisfied: numpy>=1.15.0 in /opt/conda/lib/python3.7/site-
packages (from spacy>=2.3.0->en-core-sci-md==0.2.5) (1.19.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (1.0.9)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (2.28.1)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (1.0.2)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (1.0.6)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/opt/conda/lib/python3.7/site-packages (from spacy>=2.3.0->en-core-sci-
md==0.2.5) (3.0.8)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from catalogue<1.1.0,>=0.0.7->spacy>=2.3.0->en-core-sci-md==0.2.5)
(3.10.0)
Requirement already satisfied: typing-extensions>=3.6.4 in
/opt/conda/lib/python3.7/site-packages (from
catalogue<1.1.0,>=0.0.7->spacy>=2.3.0->en-core-sci-md==0.2.5) (3.10.0.2)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.3.0->en-core-sci-md==0.2.5) (2022.9.24)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.3.0->en-core-sci-md==0.2.5) (1.26.11)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.7/site-
packages (from requests<3.0.0,>=2.13.0->spacy>=2.3.0->en-core-sci-md==0.2.5)
(3.4)
Requirement already satisfied: charset-normalizer<3,>=2 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0,>=2.13.0->spacy>=2.3.0->en-core-sci-md==0.2.5) (2.1.1)
Collecting en_core_web_sm==2.3.1
```

```
  Using cached en_core_web_sm-2.3.1-py3-none-any.whl
Requirement already satisfied: spacy<2.4.0,>=2.3.0 in
/opt/conda/lib/python3.7/site-packages (from en_core_web_sm==2.3.1) (2.3.7)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (0.7.9)
Requirement already satisfied: thinc<7.5.0,>=7.4.1 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (7.4.6)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.7/site-
packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (59.8.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (4.64.1)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (3.0.8)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (0.10.1)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (1.0.2)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (1.0.6)
Requirement already satisfied: numpy>=1.15.0 in /opt/conda/lib/python3.7/site-
packages (from spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (1.19.5)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (1.1.3)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (2.0.7)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (1.0.9)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in
/opt/conda/lib/python3.7/site-packages (from
spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (2.28.1)
Requirement already satisfied: typing-extensions>=3.6.4 in
/opt/conda/lib/python3.7/site-packages (from
catalogue<1.1.0,>=0.0.7->spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (3.10.0.2)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from
catalogue<1.1.0,>=0.0.7->spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (3.10.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from
```

```
requests<3.0.0,>=2.13.0->spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (1.26.11)
Requirement already satisfied: charset-normalizer<3,>=2 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0,>=2.13.0->spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (2.1.1)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.7/site-
packages (from
requests<3.0.0,>=2.13.0->spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0,>=2.13.0->spacy<2.4.0,>=2.3.0->en_core_web_sm==2.3.1) (2022.9.24)
 Download and installation successful
You can now load the model via spacy.load('en_core_web_sm')
```

Change `ROOT` to your path.

[2]:
```
import os
import tensorflow as tf
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

import tqdm
import pandas as pd
import random
import pickle

import readmission_utils

from tensorflow.keras.preprocessing.text import text_to_word_sequence
from tensorflow.keras.utils import to_categorical

ROOT = "/home/jupyter/cs271_assign2/ROOT"  # Put your root path here"
NUM_NS = 4  # number of negative samples in Word2Vec model
VOCAB_SIZE = 500  # numer of most common words to index in language models
tf.keras.backend.set_floatx("float32")
```

1.1 Preprocessing text data and visualization

Execute the code in the next cell, which will take about 20mins the first time you run it. It will save its results to a file in `ROOT/saved_data/texts_to_labels_1000.pkl`.

If the file already exists then calling the function will just load the results. We'll explain what it's doing later.

[3]:
```
notes, labels_admission = readmission_utils.get_notes_and_labels(ROOT, 1000)
```

```
Found file /home/jupyter/cs271_assign2/ROOT/saved_data/texts_to_labels_1000.pkl,
loading
```

Q2.2 admissions database

As in part 1, let's briefly look at the underlying data tables. Our task will be to predict hospital readmission, so we're interested in the file `ADMISSION.csv` which is in `ROOT/mimic_database`. Load the table to a dataframe and display it. One column is "INSURANCE" and the values are one of five insurance categories. Print counts of how many rows are in each insurance category (hint: use `groupby()` again).

[4]:
```
# YOUR CODE HERE #
path = "/home/jupyter/cs271_assign2/ROOT/mimic_database/ADMISSIONS.csv"
admissions = pd.read_csv(path)
display(admissions)
insurance = admissions.groupby(['INSURANCE']).agg("count")["SUBJECT_ID"]
for i in np.arange(len(insurance)):
    print("{} insurance counts are = {}".format(insurance.index[i], insurance.
 ↪values[i]))
# END CODE #
```

```
       ROW_ID  SUBJECT_ID  HADM_ID            ADMITTIME            DISCHTIME  \
0          21          22   165315  2196-04-09 12:26:00  2196-04-10 15:54:00
1          22          23   152223  2153-09-03 07:15:00  2153-09-08 19:10:00
2          23          23   124321  2157-10-18 19:34:00  2157-10-25 14:00:00
3          24          24   161859  2139-06-06 16:14:00  2139-06-09 12:48:00
4          25          25   129635  2160-11-02 02:06:00  2160-11-05 14:55:00
…         …          …        …                  …                  …
58971   58594       98800   191113  2131-03-30 21:13:00  2131-04-02 15:02:00
58972   58595       98802   101071  2151-03-05 20:00:00  2151-03-06 09:10:00
58973   58596       98805   122631  2200-09-12 07:15:00  2200-09-20 12:08:00
58974   58597       98813   170407  2128-11-11 02:29:00  2128-12-22 13:11:00
58975   58598       98813   190264  2131-10-25 03:09:00  2131-10-26 17:44:00

                 DEATHTIME ADMISSION_TYPE         ADMISSION_LOCATION  \
0                      NaN      EMERGENCY       EMERGENCY ROOM ADMIT
1                      NaN       ELECTIVE  PHYS REFERRAL/NORMAL DELI
2                      NaN      EMERGENCY  TRANSFER FROM HOSP/EXTRAM
3                      NaN      EMERGENCY  TRANSFER FROM HOSP/EXTRAM
4                      NaN      EMERGENCY       EMERGENCY ROOM ADMIT
…                      …            …                       …
58971                  NaN      EMERGENCY  CLINIC REFERRAL/PREMATURE
58972  2151-03-06 09:10:00      EMERGENCY  CLINIC REFERRAL/PREMATURE
58973                  NaN       ELECTIVE  PHYS REFERRAL/NORMAL DELI
58974                  NaN      EMERGENCY       EMERGENCY ROOM ADMIT
58975                  NaN      EMERGENCY  CLINIC REFERRAL/PREMATURE

             DISCHARGE_LOCATION INSURANCE LANGUAGE           RELIGION  \
0      DISC-TRAN CANCER/CHLDRN H   Private      NaN       UNOBTAINABLE
1               HOME HEALTH CARE  Medicare      NaN           CATHOLIC
2               HOME HEALTH CARE  Medicare     ENGL           CATHOLIC
3                           HOME   Private      NaN  PROTESTANT QUAKER
4                           HOME   Private      NaN       UNOBTAINABLE
```

```
…                          …        …       …                 …
58971                   HOME   Private     ENGL      NOT SPECIFIED
58972            DEAD/EXPIRED  Medicare     ENGL           CATHOLIC
58973        HOME HEALTH CARE   Private     ENGL      NOT SPECIFIED
58974                     SNF   Private     ENGL           CATHOLIC
58975                    HOME   Private     ENGL           CATHOLIC

      MARITAL_STATUS ETHNICITY            EDREGTIME            EDOUTTIME  \
0            MARRIED     WHITE  2196-04-09 10:06:00  2196-04-09 13:24:00
1            MARRIED     WHITE                  NaN                  NaN
2            MARRIED     WHITE                  NaN                  NaN
3             SINGLE     WHITE                  NaN                  NaN
4            MARRIED     WHITE  2160-11-02 01:01:00  2160-11-02 04:27:00
…                …         …                    …                    …
58971         SINGLE     WHITE  2131-03-30 19:44:00  2131-03-30 22:41:00
58972        WIDOWED     WHITE  2151-03-05 17:23:00  2151-03-05 21:06:00
58973        MARRIED     WHITE                  NaN                  NaN
58974        MARRIED     WHITE  2128-11-10 23:48:00  2128-11-11 03:16:00
58975        MARRIED     WHITE  2131-10-25 00:08:00  2131-10-25 04:35:00

                                                 DIAGNOSIS  \
0                                  BENZODIAZEPINE OVERDOSE
1        CORONARY ARTERY DISEASE\CORONARY ARTERY BYPASS…
2                                               BRAIN MASS
3                            INTERIOR MYOCARDIAL INFARCTION
4                                  ACUTE CORONARY SYNDROME
…                                                        …
58971                                               TRAUMA
58972                                                  SAH
58973                                    RENAL CANCER/SDA
58974                                             S/P FALL
58975                              INTRACRANIAL HEMORRHAGE

       HOSPITAL_EXPIRE_FLAG  HAS_CHARTEVENTS_DATA
0                         0                     1
1                         0                     1
2                         0                     1
3                         0                     1
4                         0                     1
…                       …                     …
58971                     0                     1
58972                     1                     1
58973                     0                     1
58974                     0                     0
58975                     0                     1

[58976 rows x 19 columns]
```

```
Government insurance counts are = 1783
Medicaid insurance counts are = 5785
Medicare insurance counts are = 28215
Private insurance counts are = 22582
Self Pay insurance counts are = 611
```

Q2.3 events text database

We'll be using the raw clinician notes from `NOTEEVENTS.CSV`, also in `ROOT/mimic_database`. This is a big file, so load in just the first 10 rows, and print them. You might notice that all 'CATEGORY' columns are type 'Discharge summary'.

Then print the full text of the first row, (the 'TEXT' column). Note that this should be over 10 lines of visible text; if you don't select the entry correctly then you may see an abbreviated version.

[5]:
```python
# YOUR CODE HERE #
path = "/home/jupyter/cs271_assign2/ROOT/mimic_database/NOTEEVENTS.csv"
noteevents = pd.read_csv(path, nrows = 10)
display(noteevents)
print(noteevents['TEXT'][0])
# END CODE #
```

```
   ROW_ID  SUBJECT_ID  HADM_ID   CHARTDATE  CHARTTIME  STORETIME  \
0     174       22532   167853  2151-08-04        NaN        NaN
1     175       13702   107527  2118-06-14        NaN        NaN
2     176       13702   167118  2119-05-25        NaN        NaN
3     177       13702   196489  2124-08-18        NaN        NaN
4     178       26880   135453  2162-03-25        NaN        NaN
5     179       53181   170490  2172-03-08        NaN        NaN
6     180       20646   134727  2112-12-10        NaN        NaN
7     181       42130   114236  2150-03-01        NaN        NaN
8     182       56174   163469  2118-08-12        NaN        NaN
9     183       56174   189681  2118-12-09        NaN        NaN

            CATEGORY DESCRIPTION  CGID  ISERROR  \
0  Discharge summary      Report   NaN      NaN
1  Discharge summary      Report   NaN      NaN
2  Discharge summary      Report   NaN      NaN
3  Discharge summary      Report   NaN      NaN
4  Discharge summary      Report   NaN      NaN
5  Discharge summary      Report   NaN      NaN
6  Discharge summary      Report   NaN      NaN
7  Discharge summary      Report   NaN      NaN
8  Discharge summary      Report   NaN      NaN
9  Discharge summary      Report   NaN      NaN

                                                TEXT
0  Admission Date:  [**2151-7-16**]       Dischar…
1  Admission Date:  [**2118-6-2**]       Discharg…
```

```
2  Admission Date:  [**2119-5-4**]             D…
3  Admission Date:  [**2124-7-21**]             …
4  Admission Date:  [**2162-3-3**]             D…
5  Admission Date:  [**2172-3-5**]             D…
6  Admission Date:  [**2112-12-8**]             …
7  Admission Date:  [**2150-2-25**]             …
8  Admission Date:  [**2118-8-10**]             …
9  Admission Date:  [**2118-12-7**]             …
```

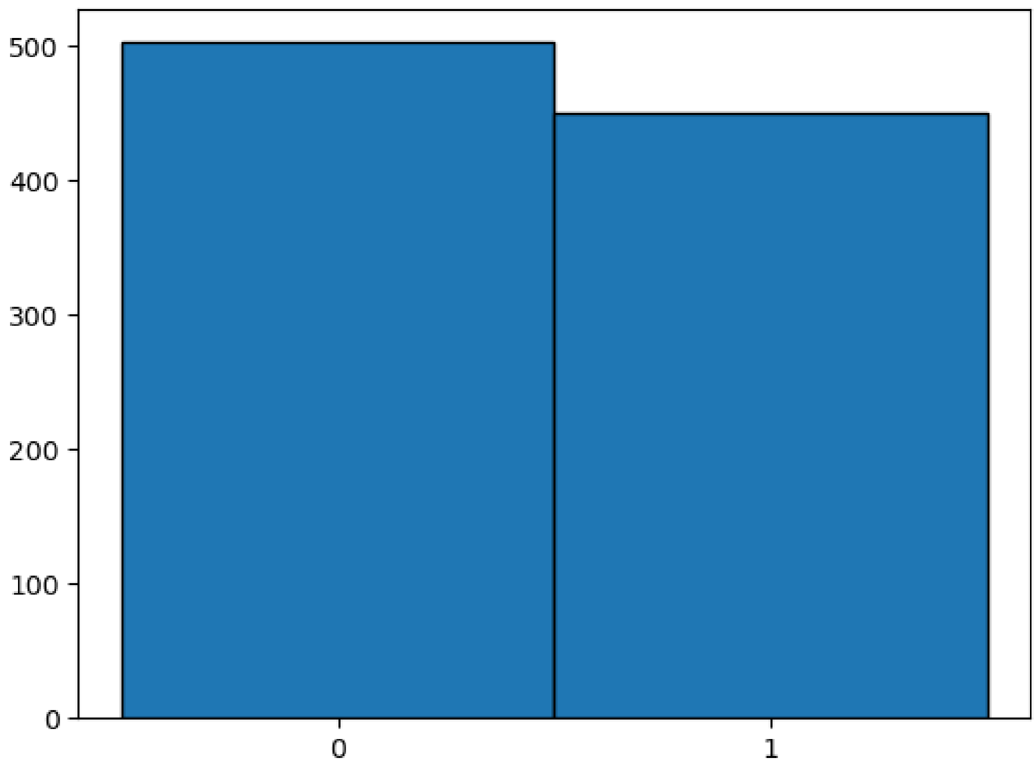
```
Admission Date:  [**2151-7-16**]       Discharge Date:  [**2151-8-4**]


Service:
ADDENDUM:

RADIOLOGIC STUDIES:  Radiologic studies also included a chest
CT, which confirmed cavitary lesions in the left lung apex
consistent with infectious process/tuberculosis.  This also
moderate-sized left pleural effusion.

HEAD CT:  Head CT showed no intracranial hemorrhage or mass
effect, but old infarction consistent with past medical
history.

ABDOMINAL CT:  Abdominal CT showed lesions of
T10 and sacrum most likely secondary to osteoporosis. These can
be followed by repeat imaging as an outpatient.



                            [**First Name8 (NamePattern2) **] [**First Name4
(NamePattern1) 1775**] [**Last Name (NamePattern1) **], M.D.  [**MD Number(1)
1776**]

Dictated By:[**Hospital 1807**]
MEDQUIST36

D:  [**2151-8-5**]  12:11
T:  [**2151-8-5**]  12:21
JOB#:  [**Job Number 1808**]
```

At the start of this assignment you ran `readmission_utils.get_notes_and_labels(ROOT)`. It did the following. - Sampled about 1000 patient admissions from `ADMISSIONS.csv` and extracted their discharge summary text from `NOTEEVENTS.csv`. - For the admission `i`: - `notes[i]` is the discharge summary text. - `labels[i]` is a `1` if that patient was readmitted after 30 days, and `0` otherwise. - The entries are randomly shuffled

Use the next code cell to compute the amount of class imbalance in the sampled dataset. You can just print the counts of labels, or show them as a histrogram.

[6]:
```python
# YOUR CODE HERE #
plt.hist(labels_admission, bins=[-.5,.5,1.5], ec="k");
plt.xticks((0,1));
unique = np.unique(labels_admission, return_counts=True)
print("There number of readmitted patients are {}. The number of non readmitted␣
↪patients are {}.".format(unique[1][1], unique[1][0]))
# END CODE #
```

```
There number of readmitted patients are 449. The number of non readmitted
patients are 502.
```



1.1.1 Word embeddings

Q2.4 tokenizers

We now want to learn a word embedding model, so that we can convert the words in `notes` to vectors that can be fed into a deep learning model.

The first step is to tokenize the notes. Execute the code in the next cell. You can read about what it's doing here.

```
[7]: vocab_size = VOCAB_SIZE
tokenizer = tf.keras.preprocessing.text.Tokenizer(
    num_words=vocab_size, oov_token="<unk>", filters='!"#$%&()*+.,:;=?
 ↪@[\]^_`{|}~/ \n'
)
tokenizer.fit_on_texts(notes)
notes_seq = tokenizer.texts_to_sequences(notes)
```

```
[ ]:
```

Notice that the `tokenizer` is only going to index the 500 most common words, and set the remainder to `<unk>`. Try printing the result of `tokenizer.index_word` and `tokenizer.word_index`. (Please do not actually print these dictionaries when you submit the assignment; they print ~1000 lines of text).

Explain the content of `notes_seq`, and how it relates to `notes`.

1.1.2 Written answer: Notes_seq is a list of sequences where each text in notes is transformed to a sequence of integers; i.e. takes each word in notes and replaces it with the corresponding integer value. In tokenizer.fit_on_texts, we create internal vocabulary based on a list of texts - an index dictionary so every word gets a unique integer value. Tokenizer.texts_to_sequences uses this dictionary to transform the given notes to sequences of integer values.

After running `tokenizer.fit_on_texts(notes)`, the `tokenizer` object stores the word counts that are in `notes` . Complete the below function to return an array of words with an array of their word counts. The arrays do not need to be sorted. Then execute the code to print the 50 most common words.

```
[8]: def get_words_and_counts(tokenizer):
    """
    Return an array of `words` and an array of their `counts` for the dataset␣
 ↪fitted to
    Keras `tokenizer` object, so that words[i] appear counts[i] times. The␣
 ↪array does
    not need to be sorted.

    Parameters:
    tokenizer (tf.keras.preprocessing.text.Tokenizer), prefitted tokenizer.

    Returns:
    vocab_words_sorted (np.array(str)) of words trained on `tokenizer`.
    vocab_words_counts_sorted (np.array(int)) word counts so that counts[i] is␣
 ↪count of words[i]
    """
    # YOUR CODE HERE
    word_counts = tokenizer.word_counts
    vocab_words = np.array(list(word_counts.keys()))
```

```python
    vocab_words_counts = np.array(list(word_counts.values()))

    # END CODE #
    return vocab_words, vocab_words_counts


# Provided code for printing the 50 most common words #
n = 50
vocab_words, vocab_words_counts = get_words_and_counts(tokenizer)
indx_sorted = np.argsort(np.array(vocab_words_counts))[::-1]
vocab_words_sorted, vocab_words_counts_sorted = (
    np.array(vocab_words)[indx_sorted],
    np.array(vocab_words_counts)[indx_sorted],
)
print("Cnt\t Word")
for i in range(n):
    print(f"{vocab_words_counts_sorted[i]}\t{repr(vocab_words_sorted[i])}")
```

```
Cnt      Word
36576   'the'
30778   'and'
26814   'to'
25842   'of'
25136   'was'
18829   'with'
17399   'a'
16993   'on'
15605   '1'
14634   'in'
13325   'for'
12865   '2'
11495   'no'
11416   'mg'
10556   'patient'
10298   'tablet'
9949    'is'
8592    'he'
8210    'blood'
8014    'po'
7915    '5'
7758    'at'
7615    '3'
7178    'name'
7031    'she'
6906    'as'
6803    'or'
6713    'discharge'
```

```
6666    'daily'
6554    'day'
6485    '4'
6361    'his'
6290    'sig'
6201    'one'
5782    '-'
5718    'history'
5438    '0'
5325    'her'
5257    '6'
5093    'left'
5081    'last'
4704    'were'
4379    's'
4355    'had'
4248    '7'
4247    'by'
4247    'be'
4158    '8'
4083    'admission'
4069    'right'
```

1.2 Word2Vec

We will now implement the Word2Vec skip-gram model. This is similar to the regular skip-gram model, but with negative sampling and subsampling (which we'll explain soon). Some background resources you may be interested in are the original paper, Distributed Representations of Words and Phrases and their Compositionality, and this blog post, Illustrated Word2Vec.

Here is a high level description of the Word2Vec model: - Take a 'target_word', one 'similar' (positive context) word, and 4 'dissimilar' (negative context) words. These words are represented as integers. - Embed each word into a vector representation (e.g. a 32-dim vector). This component is the *word embedding layer*. - Then, taking the word embeddings, predict which of the 5 context words is the 'positive context' word.

So we show the model the target word: > `[target_word]`

And 5 context words:

```
[pos_context_word, neg_context_word_1, neg_context_word_2,
neg_context_word_3, neg_context_word_4]
```

Since the positive context is at index 0, the label we train the model to predict is:

```
[1,0,0,0,0]
```

After training, we take the word embedding model and use it for other nlp tasks, like readmission prediction.

Q2.5 poitive context words

A positive `context_word` for a `target_word` is one of the previous 2 words or the next 2 words. For example, if our sentence is:

> Started on ceftriaxone and azithromycin in the ED, continued in the MICU.

And the `target_word=ceftriaxone`, then the positive context words are `started`, `on`, `and`, and `azithromycin`.

However the following are NOT positive context words because they are more than 2 words away from the target word: `ED` and `MICU`.

The idea motivating the skip-gram model is that words with similar contexts should have similar word embeddings, and we are going to enforce this when we train the Word2Vec model. Based on the examples just given of what are NOT examples of positive context words, what is one weakness of the skip-gram model for learning word embeddings?

1.2.1 Written answer: There are a few weaknesses of the skip-gram model for learning word embeddings. First, in our example above, 'ED' and 'MICU' are very temporally important context words for the target word. However, since they are more than the threshold distance away, they will not be considered. This strict thresholding could result in important context words being missed. Second, the skip-gram model cannot capture polysemy because it tends to represent a word as a single vector. For example, skip-gram cannot distinguish bank as river bank and bank as a financial institution. Lastly, skip-gram ignores the morphological information of words. Thus, words with the same morphological backbone (such as joyful and joyfullness) are considered as completely separate entities.

Q2.6 defining skipgram contexts

We'll build the dataset over a few functions. Note that we're working with tokenized words from `notes_seq`, so all the data will be integers instead of strings.

In the next cell, complete the function `build_target_contexts`. You should iterate over each note, and then iterate over each word token in each note to create an array of `targets` and an array of `positive_contexts` for those targets. E.g. suppose the start of `notes_seq` is this: > `notes_seq[ [1,6,3,4,7,8,6,...], [...], [...], ...]`

Then one valid data point will be `targets[i]=4` and `positive_contexts[i]=[6,3,7,8]`.

We set a 2-length context window, so any target can have between 0 and 4 positive context words (some targets will be at the start or end of the sequence and so they have fewer than 4 context words). If a target has fewer than 4 context words, then do not add it to the dataset. This is a simplification that shouldn't affect the dataset too much since the individual notes are long.

Note also that if the word `7` appears 100 times in the text, then it will apear 100 times in `targets` as well (unless it's omitted for having fewer than 4 context words).

The expected shape for `targets` is `(n,)`, and for `positive_contexts` is `(n,4)`. This function can run in under 20 seconds when `len(notes)<1000`.

[9]:
```
def build_target_contexts(notes_seq, context_window=2):
    """
```

```
    Given a `notes_seq`, a list of lists of tokens, add each valid token to a
    numpy array `targets`, and add its positive context window to numpy array
    `positive_contexts`. The contexts are with a window `context_window` forward
    and `context_window` back.

    All words are tokenized (represented by ints).
    E.g. for the sequence `[1,5,2,8,3,0,7]`, with context_window=2
    One returned array would be
        targets[i] = 8
        positive_contexts[i] = [5,2,3,0]

    Invalid tokens:
    In the above example, if the target is near the edges, the context vector␣
↪will be
    smaller than 4, e.g.
        targets[i] = 7
        positive_contexts[i] = [3,0]
    In this case, where len(context_window)!=2*context_window, we omit the data␣
↪point.

    Args
    notes_seq (List[List[int]]): A list of note representations, so that␣
↪notes_seq[i]
        is note i, represented by an list of token ids (which are ints).
    context_window (int): the word-distance back and forward that is still in␣
↪context.

    Returns:
    targets (np.array[int]): indices for the target words.
    positive_contexts (np.array[int,int]): array of array of context words. The␣
↪shape
        will be (n,2*context_window).
    """
    targets, positive_contexts = [], []
    for note in tqdm.tqdm(notes_seq):
        # YOUR CODE HERE #
        targets += note[2:-2]
        positive_contexts += [[note[i-2], note[i-1], note[i+1], note[i+2]] for␣
↪i in np.arange(2,len(note) - 2)]

        # END CODE #
    assert len(targets) == len(positive_contexts)
    return np.array(targets), np.array(positive_contexts)


# Run build_target
```

```
targets, positive_contexts = build_target_contexts(notes_seq, 2)
print(targets.shape, positive_contexts.shape, "\n")

# To verify results make sense, print the first tokens of the first note, and
 ↪the first set of targets and contexts #
# The targets and contexts should be the first valid ngrams of the printed note
 ↪#
print("Start of the dataset:")
print(notes_seq[0][:20])
print(f"\nTargets\t\tPositive contexts")
for i in range(10):
    print(f"{targets[i]}\t\t{positive_contexts[i]}")
```

```
100%|          | 951/951 [00:03<00:00, 271.10it/s]

(1542542,) (1542542, 4)

Start of the dataset:
[50, 61, 1, 29, 61, 1, 61, 5, 323, 1, 320, 150, 120, 1, 36, 64, 1, 111, 47, 1]

Targets         Positive contexts
1               [50 61 29 61]
29              [61  1 61  1]
61              [ 1 29  1 61]
1               [29 61 61  5]
61              [ 61   1   5 323]
5               [  1  61 323   1]
323             [ 61   5   1 320]
1               [  5 323 320 150]
320             [323   1 150 120]
150             [  1 320 120   1]
```

Q2.7 subsampling

In Q2.3, we saw a very high frequency of simple words like 'the', 'and', and 'to'. One trick used in Word2Vec is 'subsampling'; we want to sample more frequent words less often. In the below cell, we provide a function that does subsampling for you. We'll explain how it works, and then ask a question.

The `do_subsampling` function checks the target words in the dataset, and removes words at random, but it removes frequent words with a higher probability. Here is how it works: - It create a `sampling_table` (see the keras API see documentation). It has size `VOCAB_SIZE`, so it returns a VOCAB_SIZE-element array containing probabilities. - The ith most common word should have a sampling probability of `sampling_table[i]`. For example `sampling_table[0]=0.00315` is the most common word and is sampled 0.3% of the time, while `sampling_table[-1]=0.184` is the least common word and is sampled 18% of the time. Note that these numbers depend on `sampling factor` argument which is a chosen hyperparameter that could be tuned. - For each word, look up its sampling rate from `sampling_table`. It turns out that the Keras tokenizer indexes words in order of decreasing frequency, so the sampling rate for word `token_id` will be `sampling_table[token_id]`. - Remove words at random according to its sampling rate.

Run the code and then answer the written question.

[10]:
```python
### provided code for subsampling ###
def do_subsampling(targets, positive_contexts, vocab_size=500,␣
↪sampling_factor=1e-05):
    """
    Given a list of targets and contexts output from build_target_contexts,␣
↪reduce
    the size by removing words with a probability from
    tf.keras.preprocessing.sequence.make_sampling_table.

    Args:
    targets (np.array[int]): same as output of build_target_contexts.
    positive_contexts (np.array[int,int]): same as output of␣
↪build_target_contexts.

    Returns:
    targets_subsampled (np.array[int]): reduced version of targets after␣
↪subsampling
    positive_contexts_subsampled (np.array[int,int]): reduced version of␣
↪positive_contexts after subsampling
    """
    # generate sampling table
    sampling_table = tf.keras.preprocessing.sequence.make_sampling_table(
        vocab_size, sampling_factor=sampling_factor
    )
    # lookup sampling rates, using the fact that  get sample rates
    sampling_rates = sampling_table[targets]
    # generate random numbers to compare to the sampling rates
    random_nums = np.random.sample(len(sampling_rates))
    # generate True/False for whether to keep this sample
    do_sample = random_nums < sampling_rates
    # create new array having filtered some words
    targets_subsampled = targets[do_sample]
    positive_contexts_subsampled = positive_contexts[do_sample]
    return targets_subsampled, positive_contexts_subsampled


### provided code for subsampling ###

# run subsampling
print(f"Original dataset shapes          {targets.shape}, {positive_contexts.
↪shape}")
targets_subsampled, positive_contexts_subsampled = do_subsampling(
    targets, positive_contexts, vocab_size=VOCAB_SIZE
```

```
)
print(
    f"Dataset shapes after subsampling {targets_subsampled.shape},␣
↪{positive_contexts_subsampled.shape}"
)
```

```
Original dataset shapes          (1542542,), (1542542, 4)
Dataset shapes after subsampling (64206,), (64206, 4)
```

According to the original paper, what are the benefits of subsampling?

1.2.2 Written answer: According to the original paper, subsampling of frequent words during training results in significant speedup (around 2x-10x), and improves the accuracy of the representations of less frequent words.

Q2.8 Negative Sampling

Before Q2.4, we explained how Word2Vec works. Recall that we need to give the model a target word, a positive context word, and 4 negative context words. The model's task is to predict which word is the positive context word.

Our next step is to generate the negative context words. Firstly we provide a function for generating negative samples. Run the next cell and look at the example usage to make sure you understand what it's doing.

[11]:
```
def get_negative_samples(target, postive_context, num_ns=4, vocab_size=500):
    """
    Given a target word index and a list of positive context integers, randomly
    sample new integers not in `target` or `postive_context`. Generate `num_ns`␣
↪samples.

    Args
    target (int): target int that should not be in `negative_context`
    postive_context (List(int)): positive int that should not be in␣
↪`negative_context`
    num_ns (int): number of negative samples to return.
    vocab_size (int): size of vocabulary indexed by ints [0,vocab_size].

    Returns:
    negative_context (np.array[int]). Negative context tokens shape (num_ns,).
    """
    neg_samples_candidates = list(
        set(np.arange(vocab_size)) - set(postive_context) - set([target])
    )
    negative_context = np.random.choice(
        neg_samples_candidates, size=num_ns, replace=False
    )
    return negative_context
```

```
target_test = 5
positive_context_test = [1, 2, 8, 9]
vocab_size_test = 10
num_ns_test = 4
print(
    f"Test generating 10 sets of negative sampling with target word␣
 ↪{target_test}, vocab_size {vocab_size_test}, positive context␣
 ↪{positive_context_test}\n"
)
for i in range(10):
    print(
        get_negative_samples(
            target_test, positive_context_test, num_ns_test, vocab_size_test
        )
    )
```

```
Test generating 10 sets of negative sampling with target word 5, vocab_size 10,
positive context [1, 2, 8, 9]

[7 6 3 0]
[6 7 3 4]
[6 4 0 3]
[6 0 7 4]
[0 6 3 4]
[4 7 0 6]
[4 0 7 6]
[0 7 6 3]
[3 0 4 7]
[3 7 6 4]
```

We already have `targets_subsampled` and `positive_contexts_subsampled`. Let's now produce a third array `negative_contexts_subsampled` which will hold our negative samples.

We are storing each target with its entire conext window: > `targets[i]=8`, and `positive_contexts[i]=[5,2,3,0]`

But in the final model we'll actually want to generate 4 training samples with this, one for each context word. So we'l get samples `[8,5]`, `[8,2]`, `[8,3]`, and `[8,0]`. And for each one of these pairs we want to generate `NUM_NS=4` negative samples. Since there are 4 training pairs in each `positive_contexts[i]`, we will need to generate `4*NUM_NS=16` negative samples for each `targets[i]`.

Implement this in the next function by making use of the function `get_negative_samples`. It should run in about 1 minute.

[12]:
```
def build_target_positive_and_negative_contexts(
    targets_subsampled, positive_contexts_subsampled, num_ns=4, vocab_size=500
):
```

```
    """
    Generate negative context words for `targets_subsampled` and␣
↪`positive_contexts_subsampled`.
    Uses `get_negative_samples` method.

    Args:
    targets_subsampled (np.array[int]): same as output from `do_subsampling`.
    positive_contexts_subsampled (np.array([int,int])) same as output from␣
↪`do_subsampling`.
    num_ns (int): number of negative samples per array.
    vocab_size (int): vocab size. All all_targetes[i]<vocab_size.

    Returns:
    negative_contexts_subsampled (np.array[int,int]):  shape␣
↪(n_samples,n_p*num_ns) where
        n_p is the number of context words per target,␣
↪n_p=positive_contexts_subsampled.shape[1].
        The negative context words for the p samples.
    """
    n, n_p = positive_contexts_subsampled.shape
    negative_contexts_subsampled = np.zeros((n, n_p * num_ns))
    for i in tqdm.trange(n):
        # YOUR CODE HERE #
        negative_contexts_subsampled[i, :] =␣
↪get_negative_samples(targets_subsampled[i], positive_contexts_subsampled[i],␣
↪n_p * num_ns, vocab_size)
        # END CODE #
    return negative_contexts_subsampled


negative_contexts_subsampled = build_target_positive_and_negative_contexts(
    targets_subsampled,
    positive_contexts_subsampled,
    num_ns=NUM_NS,
    vocab_size=VOCAB_SIZE,
)
print(
    targets_subsampled.shape,
    positive_contexts_subsampled.shape,
    negative_contexts_subsampled.shape,
)  # expect (n,) (n,4) (n,16)
```

```
100%|     | 64206/64206 [00:07<00:00, 8163.72it/s]

(64206,) (64206, 4) (64206, 16)
```

The previous cell explained how each row in `targets[i]` will have 4 data points: one for each positive context. In the next cell we create the final dataset with some simple reshape operations.

Look at the shape of these arrays. They have 4 times the rows as the prvious cell. Each target has 1 positive context, and 4 negative contexts.

[13]:
```
n, n_p = positive_contexts_subsampled.shape
dataset_targets = np.reshape(np.repeat(targets_subsampled, 4, axis=None), (4 *␣
↪n, 1))
dataset_positive_contexts = np.reshape(positive_contexts_subsampled, (4 * n, 1))
dataset_negative_contexts = np.reshape(negative_contexts_subsampled, (4 * n, 4))
print(
    dataset_targets.shape,
    dataset_positive_contexts.shape,
    dataset_negative_contexts.shape,
)
```

```
(256824, 1) (256824, 1) (256824, 4)
```

Q2.9 Word2Vec word embedding layer

Execute the next cell. It combines the positive and negative context arrays into one array that will be passed to Word2Vec. The model will try to predict which of the 5 samples is the positive context.

The below code also generates ground-truth labels `labels`. Since we always put the positive context word as the first element, then all labels will be `[1,0,0,0,0]`.

[14]:
```
dataset_contexts = np.hstack((dataset_positive_contexts,␣
↪dataset_negative_contexts))
dataset_labels = np.zeros_like(dataset_contexts)
dataset_labels[:, 0] = 1

dataset = tf.data.Dataset.from_tensor_slices(
    ((dataset_targets, dataset_contexts), dataset_labels)
)
dataset = dataset.shuffle(10000).batch(1024, drop_remainder=True)
print(dataset)

print("targets example")
print(dataset_targets[:10])
print("context example (positive context in index 0)")
print(dataset_contexts[:10])
print("labels example")
print(dataset_labels[:10])
```

```
2022-11-01 04:37:02.093745: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
```

```
2022-11-01 04:37:02.103520: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-11-01 04:37:02.105336: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-11-01 04:37:02.107733: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-11-01 04:37:02.108405: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA
```

```
<BatchDataset shapes: (((1024, 1), (1024, 5)), (1024, 5)), types: ((tf.int64,
tf.float64), tf.float64)>
targets example
[[ 61]
 [ 61]
 [ 61]
 [ 61]
 [ 18]
 [ 18]
 [ 18]
 [ 18]
 [299]
 [299]]
context example (positive context in index 0)
[[  1. 120. 172.  26. 130.]
 [ 29. 457. 248.  24. 249.]
 [  1. 403. 192.  60.  22.]
 [ 61. 419. 101. 207. 269.]
 [  2. 262.   9.   4. 240.]
 [ 16. 399. 452.  93. 200.]
 [299. 253. 220. 113. 424.]
 [ 37. 326. 494. 479.  71.]
 [ 16. 130. 191. 465. 498.]
 [ 18. 236. 152. 463. 385.]]
labels example
[[1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
```

```
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]]
```

```
node zero
2022-11-01 04:37:02.110101: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-11-01 04:37:02.111789: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-11-01 04:37:02.695390: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-11-01 04:37:02.697381: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-11-01 04:37:02.699116: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-11-01 04:37:02.700808: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 13642 MB memory:  -> device:
0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
```

We have provided most of the Word2Vec model. In the next cell you need to add the model embedding layers (see Keras Embedding docs). We have different embedding functions. - Define `self.target_embedding` layer for the target. It expects 1-element arrays from `targets` (hint: use `input_length`). - Define `self.context_embedding` layer for the context (positive and negative context words). It expects 5-element arrays from `targets`.

[15]:
```
class Word2Vec(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, num_ns,␣
 ↪normalize_embeddings=False):
        super(Word2Vec, self).__init__()
        self.target_embedding = None
        self.context_embedding = None
        self.normalize_embeddings = normalize_embeddings
        # YOUR CODE HERE #
```

```
        self.target_embedding = tf.keras.layers.Embedding(vocab_size,␣
↪embedding_dim, input_length=1, name="target_embedding")
        self.context_embedding = tf.keras.layers.Embedding(vocab_size,␣
↪embedding_dim, input_length=5)
        # END CODE #

    def call(self, pair):
        target, context = pair
        target = tf.squeeze(target, axis=1)
        word_emb = self.target_embedding(target)  # word_emb: (batch, embed)
        context_emb = self.context_embedding(
            context
        )  # context_emb: (batch, context, embed)
        if self.normalize_embeddings:
            word_emb = word_emb / np.linalg.norm(word_emb, axis=1, ord=2)[:,␣
↪None]
            context_emb = (
                context_emb / np.linalg.norm(context_emb, axis=2, ord=2)[:, :,␣
↪None]
            )

        dots = tf.einsum("be,bce->bc", word_emb, context_emb)  # dots: (batch,␣
↪context)
        return dots
```

Q2.10 training Word2Vec

Create, compile and run the model. We recommend: - 100 epochs. - 32-dim word embedding dimension. - Adam optimizer with default params. - Categorical cross entropy with the following call `tf.keras.losses.CategoricalCrossentropy(from_logits=True)`

You should get accuracy >0.85.

[16]:
```
embedding_dim = 32
epochs = 100

# YOUR CODE HERE #
model_word2vec = Word2Vec(vocab_size, embedding_dim, 4)

optimizer = tf.keras.optimizers.Adam()
loss = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
metrics = ['accuracy']

model_word2vec.compile(optimizer, loss, metrics)
model_word2vec.fit(
    (dataset_targets, dataset_contexts),
    dataset_labels,
    epochs = epochs,
```

```python
)
# END CODE #
```

```
2022-11-01 04:37:03.110688: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)
```

```
Epoch 1/100
8026/8026 [==============================] - 21s 2ms/step - loss: 0.7108 -
accuracy: 0.7512
Epoch 2/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.4739 -
accuracy: 0.8308
Epoch 3/100
8026/8026 [==============================] - 20s 3ms/step - loss: 0.4141 -
accuracy: 0.8518
Epoch 4/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3865 -
accuracy: 0.8625
Epoch 5/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3698 -
accuracy: 0.8679
Epoch 6/100
8026/8026 [==============================] - 20s 3ms/step - loss: 0.3585 -
accuracy: 0.8725
Epoch 7/100
8026/8026 [==============================] - 20s 3ms/step - loss: 0.3502 -
accuracy: 0.8749
Epoch 8/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3439 -
accuracy: 0.8777
Epoch 9/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3390 -
accuracy: 0.8792
Epoch 10/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3352 -
accuracy: 0.8806
Epoch 11/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3320 -
accuracy: 0.8815
Epoch 12/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3292 -
accuracy: 0.8829
Epoch 13/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3269 -
accuracy: 0.8835
Epoch 14/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3249 -
```

```
accuracy: 0.8840
Epoch 15/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3232 -
accuracy: 0.8848
Epoch 16/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3217 -
accuracy: 0.8852
Epoch 17/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3202 -
accuracy: 0.8858
Epoch 18/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3188 -
accuracy: 0.8863
Epoch 19/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3177 -
accuracy: 0.8862
Epoch 20/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3168 -
accuracy: 0.8871
Epoch 21/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3157 -
accuracy: 0.8873
Epoch 22/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.3149 -
accuracy: 0.8877
Epoch 23/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3139 -
accuracy: 0.8878
Epoch 24/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3134 -
accuracy: 0.8878
Epoch 25/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3125 -
accuracy: 0.8884
Epoch 26/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3120 -
accuracy: 0.8884
Epoch 27/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3112 -
accuracy: 0.8888
Epoch 28/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3107 -
accuracy: 0.8886
Epoch 29/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3101 -
accuracy: 0.8890
Epoch 30/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3096 -
```

```
accuracy: 0.8888
Epoch 31/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3091 -
accuracy: 0.8888
Epoch 32/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3085 -
accuracy: 0.8894
Epoch 33/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3083 -
accuracy: 0.8893
Epoch 34/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3077 -
accuracy: 0.8894
Epoch 35/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3074 -
accuracy: 0.8896
Epoch 36/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3071 -
accuracy: 0.8897
Epoch 37/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3066 -
accuracy: 0.8897
Epoch 38/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3064 -
accuracy: 0.8902
Epoch 39/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3059 -
accuracy: 0.8902
Epoch 40/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3056 -
accuracy: 0.8901
Epoch 41/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3053 -
accuracy: 0.8903
Epoch 42/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3052 -
accuracy: 0.8905
Epoch 43/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3049 -
accuracy: 0.8905
Epoch 44/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3046 -
accuracy: 0.8905
Epoch 45/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3044 -
accuracy: 0.8907
Epoch 46/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3039 -
```

```
accuracy: 0.8908
Epoch 47/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3040 -
accuracy: 0.8907
Epoch 48/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.3038 -
accuracy: 0.8911
Epoch 49/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3034 -
accuracy: 0.8910
Epoch 50/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3031 -
accuracy: 0.8911
Epoch 51/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3030 -
accuracy: 0.8909
Epoch 52/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3028 -
accuracy: 0.8912
Epoch 53/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3026 -
accuracy: 0.8912
Epoch 54/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3025 -
accuracy: 0.8913
Epoch 55/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3022 -
accuracy: 0.8909
Epoch 56/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3020 -
accuracy: 0.8914
Epoch 57/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3018 -
accuracy: 0.8913
Epoch 58/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3017 -
accuracy: 0.8917
Epoch 59/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.3014 -
accuracy: 0.8918
Epoch 60/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3013 -
accuracy: 0.8916
Epoch 61/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3012 -
accuracy: 0.8917
Epoch 62/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3010 -
```

```
accuracy: 0.8914
Epoch 63/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3009 -
accuracy: 0.8918
Epoch 64/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.3007 -
accuracy: 0.8921
Epoch 65/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.3006 -
accuracy: 0.8921
Epoch 66/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3005 -
accuracy: 0.8916
Epoch 67/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3003 -
accuracy: 0.8922
Epoch 68/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.3003 -
accuracy: 0.8918
Epoch 69/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.3001 -
accuracy: 0.8919
Epoch 70/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.3000 -
accuracy: 0.8922
Epoch 71/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2997 -
accuracy: 0.8922
Epoch 72/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.2997 -
accuracy: 0.8922
Epoch 73/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.2996 -
accuracy: 0.8922
Epoch 74/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2995 -
accuracy: 0.8921
Epoch 75/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2993 -
accuracy: 0.8923
Epoch 76/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.2993 -
accuracy: 0.8926
Epoch 77/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2990 -
accuracy: 0.8923
Epoch 78/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2991 -
```

```
accuracy: 0.8926
Epoch 79/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.2991 -
accuracy: 0.8923
Epoch 80/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2988 -
accuracy: 0.8927
Epoch 81/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2988 -
accuracy: 0.8928
Epoch 82/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2987 -
accuracy: 0.8926
Epoch 83/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.2986 -
accuracy: 0.8924
Epoch 84/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2985 -
accuracy: 0.8927
Epoch 85/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2985 -
accuracy: 0.8923
Epoch 86/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.2984 -
accuracy: 0.8928
Epoch 87/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2982 -
accuracy: 0.8928
Epoch 88/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2981 -
accuracy: 0.8929
Epoch 89/100
8026/8026 [==============================] - 20s 2ms/step - loss: 0.2981 -
accuracy: 0.8928
Epoch 90/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2979 -
accuracy: 0.8929
Epoch 91/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2980 -
accuracy: 0.8929
Epoch 92/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.2978 -
accuracy: 0.8928
Epoch 93/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2976 -
accuracy: 0.8930
Epoch 94/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2977 -
```

```
accuracy: 0.8928
Epoch 95/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.2977 -
accuracy: 0.8929
Epoch 96/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2975 -
accuracy: 0.8934
Epoch 97/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2975 -
accuracy: 0.8930
Epoch 98/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2974 -
accuracy: 0.8930
Epoch 99/100
8026/8026 [==============================] - 19s 2ms/step - loss: 0.2973 -
accuracy: 0.8929
Epoch 100/100
8026/8026 [==============================] - 18s 2ms/step - loss: 0.2971 -
accuracy: 0.8932
```

[16]:
```
<keras.callbacks.History at 0x7f3469562e50>
```

What is the baseline accuracy for this prediction task? In other words, what 'accuracy' would you expect if we were randomly guessing predictions.

1.2.3 Written answer: The baseline accuracy would be 1/5th. The labels array is an array of 4 negative context words and 1 positive context word. Thus, if the model were to predict the 1 positive context word in 5 word array, it would have a 1/5th chance of getting it right.

Q2.11 word embeddings

Word2Vec learns to predict positive-context words from a list of positive- and negative-context words. In order to do this, Word2Vec must embed integer tokens into fixed-length vectors called 'word embeddings'. The point of doing Word2Vec is to get these embeddings, and use them for downstream prediction tasks.

Complete the following function to get a matrix that will store the word embeddings for our vocabulary. You should use the `target_embedding` layer.

[17]:
```python
def get_word_embeddings(model_word2vec, vocab_size):
    """
    Take the target word embedding layer from `model_word2vec`. Produce an␣
↪embedding
    vector for a vocabulary with size vocab_size, so that `embeddings[i]`␣
↪returns the
    word embedding vector for the ith word.

    Args:
```

```
    model_word2vec (class Word2Vec): a trained Word2Vec model.
    vocab_size (int): vocab size; the model must have been trained on this size.

    Returns:
    embeddings (np.array[float,float]): with shape (vocab_size, embedding_dim)
    """
    embeddings = None
    # YOUR CODE HERE #
    embeddings = np.zeros((vocab_size, embedding_dim))
    for i in np.arange(vocab_size):
        embedding_vector = model_word2vec.get_layer('target_embedding').
 ↪get_weights()[0][i]
        if embedding_vector is not None:
            embeddings[i] = embedding_vector

    # END CODE #
    return embeddings

#model_word2vec
embeddings = get_word_embeddings(model_word2vec, VOCAB_SIZE)
print(embeddings.shape)  # expect (VOCAB_SIZE, embedding_dim)
```

```
(500, 32)
```

Q2.12 nearest words

Word2Vec is trained so that words with similar contexts have similar word embeddings (as measured by cosine similarity).

We provide the function `find_nearest_words` below. Given a target word, it returns a string of the nearest words in the embedding space.

All you have to do for this question is add some words to the `chosen_words`, e.g. `['bleeding','pain','and']`, and then execute the code in the cell.

[18]:
```
from sklearn.metrics.pairwise import cosine_similarity

dists = cosine_similarity(embeddings, embeddings)


def find_nearest_words(word, embeddings, tokenizer):
    """
    Given
    Args:
    word (str): target word.
    embeddings (np.array[float,float]): same as output to get_word_embeddings.
    tokenizer: (tf.keras.preprocessing.text.Tokenizer) woth vocabulary␣
 ↪corresponding
```

```
        to `embeddings` st tokenizer.word_index[word]=i is the ith column of␣
↪`embeddings`.
    """
    dists = cosine_similarity(embeddings, embeddings)
    idx = tokenizer.word_index.get(word, VOCAB_SIZE)
    if idx >= VOCAB_SIZE:
        return "ERROR: NOT IN VOCAB"
    nearest = np.argsort(dists[idx])[::-1]
    print(nearest)
    nearest_words = ""
    for j in range(1, 15):
        nearest_words += tokenizer.index_word[nearest[j]] + ", "
    return nearest_words


chosen_words = None
# YOUR CODE HERE #
chosen_words = ['cough','fever','doctor']

# END CODE #

for word in chosen_words:
    nearest_words = find_nearest_words(word, embeddings, tokenizer)
    print(f"TARGET: {word}\nNEAREST: {nearest_words}")
    print("\n")
```

```
[423 219  52 356 397 290 471 224  45 207 385 358 147 490 269  75 389 391
 238 316 275 152 450 168 127 390 341 113 447 165 291 246 360 433 271  80
 446 281 480 254 218  12 492  26  28 445  37 192 272  85 221 228 395 441
 293 190 260 166 352 404  98 458 122 292  93 197 169 403 428 203 297 296
  94  27 353 408 422  57 140 274 188 329  67 285  65 418  69 205 331 179
  70 295 489  82 278 133 206 160 435 370  54 361 251 143 440  14 470 170
   5 426 493 466 311 494 350 186 467 415 327 234  23 139 115 245  63 328
 108 412 363 118 342 158 175 312   4  95 276 232 460 196 134 263 459  20
  30 223 144 444 321 255 135 388 349   1 432  19   3   8 368 375 337 235
 398 463 217 240 131 125 417 498 313 226 424 153 409  58 268 176 266 280
 457 340 227 438 334 202 172 304 308 124 120 252 146 265  83 211 112   6
 437 193 287 129 497  35 181 264 414 243  18 104 214  84 333 468 151  73
 323 442 141 222 237 367 357  31  36 256 171 101 200 187 487  92 320 351
 326  78 410 148 406 195 439 317  66 213 425 185  16 305 284  22 298 413
 362 405 220  97 354 102 472 288 477   2 381  17 465  90 299 324 330 475
 123 449   7 149  13 421  74 109 178 402  87 173 453  41 377  21 128  48
 114 279 117 136 366 194 106 159 303 309 336 374 239  40  91  49 225  33
 156 369  88 373  55 163 307 154  51 257 443 236 461 411 319 401  43  96
 335 359  50  32 499  79 380 244 250 338 346 483 289 177 394 339 248 479
  53 300 121 434 348 396  11 420 270 183 229 167  89 371 344  64 469 448
 364 231 384 182 491  34 164 116 100   9 174 209 314 145 429  44 419 386
```

```
 355 249 294  39 488 155 212 496 365 138 427  72 379  24 400 473 382 474
  56 462  86 452 204 392 486  62  59 387  47 485 431 481 119 343 393 247
 301 110  60  61 157 111 162 242 310  99  81 325  42 191  38 482 315 282
  77 273 347 132 253 142 230 378 383 261 105 495  15 456  29 258 455 322
 201 216 478 286 283 318  71  46 241 436 208 451 262 345 302 372 130 233
 210 107 199 184 407 430 476 376 306 103  10 277 150 126 464 399  68 267
 198 215 189 259  25 332 180 161 484   0 416 454  76 137]
TARGET: cough
NEAREST: abdominal, pain, fever, nausea, positive, having, denies, had, breath,
oxygen, f, without, ekg, shortness,


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


[241 161 374  68 488 282 383  99 279 267  42 132 376  28  25 334 259 476
 432  55 109 286 178  36  48  58 249 483 454  83   1 236 144 264 261  67
 381 223  33 198 167  86 318 327 380  96 301 232 464 238 346 101 354 293
```

```
 168 359 475  71 360 169 307 145   7 111  84 265 152 319 213 142 281 139
 493 412 212  78 207 253 407 458 362 330 402 188 295 441 415  75 350 208
 275  56 205 262 469 379 309 305 215 393 200 313 426 137 150  19 333 248
 470 499 365 332 367 442 251 266 147 356 184 128 209  43 285 156 130  76
 291 444 495 303 397  39 373 202 339 459 317 369 306 196 394 462 453 221
 203 187 486 143  94   4  20  23  70  61 449  47 443 247 175 268 382  90
 284 186 311 482  87 246 484  11 496 298 172 154 131  45  59 166 388 127
  41 263 490 408 368 302 190 384 210 165 473 491 271 119 385 300 277 287
 270  18 182 410 389 348  51 140 446 477 337 274 133 478 230 344 193  63
 353 363 320 421 240 355 189 494 434  34 445 371  15 296 405 220 102 452
 324   8  82 206 243 204 396 191 370 345 399 278 118 414 349 297   3 245
 417 256  60 364 113 117  92 329 107 148 433 222 338  72 179 351 390 358
 497 435  29  52 197 316 387  88 120 437 485 214 177 299 231 418 134  57
 195 480 326 242 466  54 314 255 171 122 173  26 124 395  14 192 428 431
 335 183 138  98 199 162 292 250 219 440 136 235   2 460 361 239 151  80
  35 254 404 436 377 468 273 422 492 176 391 155 126 498 227  74 234 461
   6 104  97 463  73 174  22  16 325 455  38 216 386 276 157 427 411 439
 341  77 112  65 224 304 447  12 201 336 149 448 181 457 375 347 450 160
 114 456  81 110 289 474 487 323 244 103 141 100  93 400 438   9 164 283
  64  79 294 420 331 153 194  69 366   5  49 471 467 425 121 310 272 260
 146  21 315  31 372 409 403  91 321 185  17 429 170 479  30 312 481 357
 416 108 228 424 280 252  37 343 218 398 328 257 225  53 322 406  66  44
 211 233 123 105 308  50 229  10 392 106  40 413  27 352 401 288  85 129
 159 158 378  89 226 180 423 115 116   0 430 217 163 135  95  13 269 419
 290 340 237 451  46 489 125 465  24 472  62  32 342 258]
TARGET: doctor
NEAREST: namepattern1, pcp, first, both, lung, lobe, dr, primary, lf, last, un,
facility, or, name,
```

1.2.4 Prediction with word embeddings

Q2.13 data representation for notes

Now that we have word embeddings for our notes, lets make predictions. We will provide data-generating code, and you will define the model.

The original dataset is two equally-sized lists, so that `notes[i]` is the discharge summary of visit `i`, and `labels_admission` is a `1` if there was a readmission within 30 days, and `0` otherwise. The next cell creates the input to a Keras sequence model `(batch_size, n_tokens, embedding_dim)`.

Each sequence of words can only be `n_tokens` long. Here we choose `n_tokens=512`. But the notes can be any number of tokens long. There are many strategies for choosing which word tokens to include in the note representation. We will just take the first 512 tokens from each note.

[19]:
```
def convert_notes_seq_to_embeddings(notes_seq, embeddings):
    notes_word_embeddings = []
    for i, note_seq in enumerate(notes_seq):
```

```
        note_word_embeddings = tf.gather(embeddings, indices=note_seq)
        notes_word_embeddings.append(np.array(note_word_embeddings))
    return notes_word_embeddings


notes_word_embeddings = convert_notes_seq_to_embeddings(notes_seq, embeddings)

notes_first_512_words = np.zeros((len(notes), 512, embeddings.shape[-1]))
for i in range(len(notes)):
    bs = min(512, len(notes_word_embeddings[i]))
    notes_first_512_words[i, :bs] = notes_word_embeddings[i][:bs]

print(notes_first_512_words.shape)  # expect (len(notes), 512, embedding_dim).
```

```
(951, 512, 32)
```

We chose to just use the first 512 word embeddings from each note as its representation. Suggest two other strategies we could have used,

1.2.5 Written answer: 1. We could have chose the rarest words, which typically provide more information about the patient's particular condition than common words. 2. We could take the last 512 tokens, since the end of the note is more likely to describe the patient's condition just as they are about to leave the ER.

Then execute the next cell which create train/val/test splits

[20]:
```
from sklearn.model_selection import train_test_split

DATA = notes_first_512_words
LABELS = labels_admission
X_train, X_test, y_train, y_test = train_test_split(
    notes_first_512_words, np.array(labels_admission), test_size=0.2,␣
 ↪random_state=1
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.25, random_state=1
)
print("Train ", X_train.shape, y_train.shape)
print("Val   ", X_val.shape, y_val.shape)
print("Test  ", X_test.shape, y_test.shape)
```

```
Train  (570, 512, 32) (570,)
Val    (190, 512, 32) (190,)
Test   (191, 512, 32) (191,)
```

Q2.14 training

Create a prediction model including: - 1 masking layer with mask value 0. - 1 LSTM layer that returns a single vector (instead of a sequence of vectors). - 1 dropout layer with dropout rate 0.1.

- 1 dense layer with activation whose output is a prediction.

[21]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense, Masking
```

[22]:
```
num_timesteps, num_features = X_train.shape[-2:]
num_lstm_units = 32
# YOUR CODE HERE #

model = Sequential()
model.add(tf.keras.layers.Masking(mask_value=0.))
model.add(LSTM(num_lstm_units))
model.add(Dropout(0.1))
model.add(Dense(1, activation = 'sigmoid'))

# END CODE #
```

In part 1, the LSTM layer returned a value for every element in the sequence. In this problem the LSTM layer returns only the last element. Explain why this task is different.

1.2.6 Written answer: In part 1, the LSTM was a many-to-many formulation. Now, since the LSTM layer only returns the last element, it is a many-to-one formulation. In a many-to-one problem, we have a sequence of data as input and will predict a single output. Instead of making a predictions on a word by word level, we make a total prediction at the end of the token sequence. This is similar formulation to sentiment analysis.

Compile the model with Adam, a suitable loss function, and the 'accuracy' metric. Train the model for 15 epoch.

Note that this is a very difficult model to train with the dataset that we have. Your results should show decreasing loss on the train set, but you may not see any improvement in the validation loss. The best model should achieve ~0.55 accuracy on the validation set.

[ ]:
```
# YOUR CODE HERE #
optimizer = tf.keras.optimizers.Adam()
loss = tf.keras.losses.BinaryCrossentropy()
metrics = ['accuracy']

model.compile(optimizer, loss, metrics)

epochs = 15
# YOUR CODE HERE #
model.fit(
    X_train,
    y_train,
    epochs = epochs,
    validation_data = (X_val, y_val)
)
```

```
# END CODE #
```

```
Epoch 1/15

2022-11-01 05:08:40.887435: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369]
Loaded cuDNN version 8005

18/18 [==============================] - 6s 93ms/step - loss: 0.6914 - accuracy:
0.5246 - val_loss: 0.6872 - val_accuracy: 0.5526
Epoch 2/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6926 - accuracy:
0.5333 - val_loss: 0.6913 - val_accuracy: 0.5368
Epoch 3/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6892 - accuracy:
0.5421 - val_loss: 0.6879 - val_accuracy: 0.5632
Epoch 4/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6794 - accuracy:
0.5877 - val_loss: 0.6928 - val_accuracy: 0.5158
Epoch 5/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6754 - accuracy:
0.5667 - val_loss: 0.6863 - val_accuracy: 0.5474
Epoch 6/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6651 - accuracy:
0.6035 - val_loss: 0.6895 - val_accuracy: 0.5526
Epoch 7/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6604 - accuracy:
0.6158 - val_loss: 0.6952 - val_accuracy: 0.5526
Epoch 8/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6541 - accuracy:
0.6018 - val_loss: 0.6923 - val_accuracy: 0.5579
Epoch 9/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6407 - accuracy:
0.6105 - val_loss: 0.7169 - val_accuracy: 0.5158
Epoch 10/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6369 - accuracy:
0.6474 - val_loss: 0.7032 - val_accuracy: 0.5632
Epoch 11/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6297 - accuracy:
0.6667 - val_loss: 0.7121 - val_accuracy: 0.5579
Epoch 12/15
18/18 [==============================] - 0s 20ms/step - loss: 0.6184 - accuracy:
0.6544 - val_loss: 0.7291 - val_accuracy: 0.5632
Epoch 13/15
18/18 [==============================] - 0s 20ms/step - loss: 0.5935 - accuracy:
0.6772 - val_loss: 0.7412 - val_accuracy: 0.5053
Epoch 14/15
18/18 [==============================] - 0s 20ms/step - loss: 0.5785 - accuracy:
```

```
0.7123 - val_loss: 0.7661 - val_accuracy: 0.4684
Epoch 15/15
18/18 [==============================] - 0s 21ms/step - loss: 0.5646 - accuracy:
0.7070 - val_loss: 0.7728 - val_accuracy: 0.5105
```

[ ]: <keras.callbacks.History at 0x7f3450eeec50>

Best checkpoint validation accuracy is 0.5684.

Q2.13

Suggest 2 reasons why the validation results were poor when we trained this model on this dataset.

1.2.7 Written answer: 1. We could be overfitting to the nuances of the sentences given in the training set, leading to poor validation accuracy. 2. Our method of choosing which word tokens to include could be sub-optimal (the first 512 words may not provide the proper context we need for prediction).

1.3