A2_part1_LSTM_on_EHR_structured

October 30, 2021

1 Assignment 2 - clinical prediction with LSTMs on structured MIMIC-III data

The Intensive Care Unit (ICU) treats, on estimate, 55,000 patients per day. ICU patients have an average length of stay of 3.8 days with a mortality rate of 10-29% soucre. Furthermore, monitoring patients in ICU rooms requires keeping track of tremendous amounts of real-time information, and much of it is logged and stored in electronic health record (EHR) systems. Information overload can be seen as a huge barrier to safe and efficient healthcare delivery. As such, there has been much work on exploring computer assisted diagnostic (CAD) systems to predict clinical outcomes from these data sources.

In the first part of this assignment, you'll be working with structured data measurements (e.g. heart rate, glucose levels, central venous pressure) to make predictions of - sepsis - myocardial infarction (MI) - vancomycin antibiotic administration

over two week patient ICU courses. We'll be running a simplified version of the models in An attention based deep learning model of clinical events in the intensive care unit. (In parts 2 & 3 of this assignment you'll use unstructurd clinical text from discharge summaries.)

Q1.1 clinical applications of prediction models

Many researchers work on the problem of predicting sepsis. Briefly explain how a sepsis prediction model can improve clinical outcomes for patients.

Written answer: Accurate early diagnosis of sepsis can reduce the risk of adverse patient outcomes from severe sepsis and septic shock. By identifying systemic inflammation as a sign of infection and detecting possible organ dysfunction, we can begin to recognize early signs of severe sepsis. With advanced warning of impending sepsis onset, clinicians can implement early intervention treatment to drastically improve clinical outcomes for patients.

1.1 MIMIC-III Data Preprocessing & Visualization

In the below cell, define the path to `ROOT`. This is where all assignment 2 data will be placed. E.g. you could put it in the same directory as the notebook.

Go to https://physionet.org/sign-dua/mimiciii/1.4/ and accept the MIMIC-III data use agreement.

Navigate to `ROOT` in your terminal, and then download the MIMIC-III dataset by executing the following commands (replace username with your physionet username). This will create a directory in ROOT called `mimic_database/`.

`wget -r -N -c -np --user <username> --ask-password https://physionet.org/files/mimiciii/1.4/` (2-5 minutes)

`mkdir -p mimic_database && mv physionet.org/files/mimiciii/1.4/*.csv.gz mimic_database/ && rm -rf physionet.org/ && cd mimic_database && gunzip *.gz && echo 'Succes!' || echo 'Failure'` (5-10 mins)

If everything worked, the final output should read `Success!`, and `mimic_database/` should contain some csv files.

[1]:
```python
import pickle
import math
import re
import csv
import concurrent.futures
import os
from functools import reduce
import pathlib
import pickle

from operator import add
import pandas as pd
import numpy as np

import gc
from time import time
import math
import pickle
import pathlib
import matplotlib.pyplot as plt

import tensorflow as tf
tf.keras.backend.set_floatx('float64')

# packages from current directory
import parser_utils
import data_utils

# config
tf.keras.backend.set_floatx('float32')

ROOT = "/home/marchuo/assign2"   # Put your root path here
```

(~40 mins) Run the next cell after setting `DO_PARSING` and `DO_BUILD_DATASETS` to True. It's slow, but only needs to be run once. It will create files in `ROOT/mapped_events/`, and in `ROOT/saved_data`, and we'll explain what it's doing a bit later.

Once it runs successfully, set `DO_PARSING` and `DO_BUILD_DATASETS` to False.

[2]:
```
DO_PARSING=False
DO_BUILD_DATASETS=False

if DO_PARSING: parser_utils.do_all_parsing(ROOT, verbose=1)
if DO_BUILD_DATASETS: data_utils.build_seq_datasets(ROOT)
```

Q1.2 mimic database exploration I

Let's get a better understanding of the MIMIC-III database. Here is the documentation. The 'Data Description' section is especially useful. When you ran the data dowload commands at the start of this notebook (the command starting with `wget`), a directory was created in ROOT called `mimic_databse/`. This is contains the MIMIC csv files.

First load the `PATIENTS.csv` file and display the results to screen (hint: use Pandas to load the csv's to a DataFrame; hint 2: the function `display(df)` prints the dataframes nicely).

[3]:
```
# YOUR CODE HERE #
mimic_database = "/home/marchuo/assign2/mimic_database/PATIENTS.csv"
patients_df = pd.read_csv(mimic_database)
display(patients_df)
# END CODE #
```

```
       ROW_ID  SUBJECT_ID GENDER                  DOB                  DOD  \
0         234         249      F  2075-03-13 00:00:00                  NaN
1         235         250      F  2164-12-27 00:00:00  2188-11-22 00:00:00
2         236         251      M  2090-03-15 00:00:00                  NaN
3         237         252      M  2078-03-06 00:00:00                  NaN
4         238         253      F  2089-11-26 00:00:00                  NaN
…         …          …    …                   …                   …
46515   31840       44089      M  2026-05-25 00:00:00                  NaN
46516   31841       44115      F  2124-07-27 00:00:00                  NaN
46517   31842       44123      F  2049-11-26 00:00:00  2135-01-12 00:00:00
46518   31843       44126      F  2076-07-25 00:00:00                  NaN
46519   31844       44128      M  2098-07-25 00:00:00                  NaN

                 DOD_HOSP DOD_SSN  EXPIRE_FLAG
0                     NaN     NaN            0
1     2188-11-22 00:00:00     NaN            1
2                     NaN     NaN            0
3                     NaN     NaN            0
4                     NaN     NaN            0
…                     …     …           …
46515                 NaN     NaN            0
46516                 NaN     NaN            0
46517 2135-01-12 00:00:00     NaN            1
46518                 NaN     NaN            0
46519                 NaN     NaN            0

[46520 rows x 8 columns]
```

Notice that the date of birth (DOB) and date of death (DOD) are in the future. Briefly explain why (hint: see Methods section of MIMIC documentation).

Written answer: Before data could be incorporated into the database, it has to be deidentified. In particular, the dates were deidentified by shifting the dates into the future by some random interval that is consistent with each patient to preserve intervals. This results in stays that occurred in the future between 2100 and 2200.

Q1.3 mimic database exploration II

In the next code cell, use the dataframe from PATIENTS.csv to print the following summary measurements about the dataset: - The number of total patients. - The counts of male and female patients. - The count of patients with a death on record.

(Hint: the groupby() function may be useful).

[4]:
```python
# YOUR CODE HERE #
num_patients = len(patients_df)
num_gender = patients_df.groupby(["GENDER"]).size()
num_deaths = patients_df[patients_df["EXPIRE_FLAG"] == 1]
print("Number of Patients: " + str(num_patients))
print("Number of Patient Deaths: " + str(num_deaths.shape[0]))
print(num_gender)
# END CODE #
```

```
Number of Patients: 46520
Number of Patient Deaths: 15759
GENDER
F    20399
M    26121
dtype: int64
```

Q1.4 mimic database exploration III

Let's now look at CHARTEVENTS.csv, which has one row for each recorded chart measurement (e.g. features like heart rate, glucose levels, central venous pressure). This file is 33GB so don't try to load the whole thing into memory.

Read the first 100 rows of the file CHARTEVENTS.csv and display the DataFrame in the notebook.

[5]:
```python
first_nrows = 100
# YOUR CODE HERE #
events_path = "/home/marchuo/assign2/mimic_database/CHARTEVENTS.csv"
events_df = pd.read_csv(events_path, nrows=first_nrows)
display(events_df)
# END #
```

```
   ROW_ID  SUBJECT_ID  HADM_ID  ICUSTAY_ID  ITEMID            CHARTTIME  \
0     788          36   165660      241249  223834  2134-05-12 12:00:00
1     789          36   165660      241249  223835  2134-05-12 12:00:00
2     790          36   165660      241249  224328  2134-05-12 12:00:00
3     791          36   165660      241249  224329  2134-05-12 12:00:00
```

```
4      792          36   165660      241249  224330  2134-05-12 12:00:00
..     …          …      …        …      …                  …
95     348          34   144319      290505  226873  2191-02-23 07:31:00
96     349          34   144319      290505  220210  2191-02-23 07:33:00
97     350          34   144319      290505  220045  2191-02-23 07:34:00
98     351          34   144319      290505  220179  2191-02-23 07:34:00
99     352          34   144319      290505  220180  2191-02-23 07:34:00

              STORETIME   CGID   VALUE  VALUENUM  VALUEUOM  WARNING  ERROR  \
0   2134-05-12 13:56:00  17525   15.00     15.00     L/min        0      0
1   2134-05-12 13:56:00  17525  100.00    100.00       NaN        0      0
2   2134-05-12 12:18:00  20823    0.37      0.37       NaN        0      0
3   2134-05-12 12:19:00  20823    6.00      6.00       min        0      0
4   2134-05-12 12:19:00  20823    2.50      2.50       NaN        0      0
..                  …      …      …       …         …       …     …
95  2191-02-23 07:35:00  16924    1.00      1.00       NaN        0      0
96  2191-02-23 07:45:00  17741   26.00     26.00  insp/min        0      0
97  2191-02-23 10:53:00  17741   44.00     44.00       bpm        0      0
98  2191-02-23 07:45:00  17741  135.00    135.00      mmHg        0      0
99  2191-02-23 07:45:00  17741   61.00     61.00      mmHg        0      0

    RESULTSTATUS  STOPPED
0            NaN      NaN
1            NaN      NaN
2            NaN      NaN
3            NaN      NaN
4            NaN      NaN
..           …       …
95           NaN      NaN
96           NaN      NaN
97           NaN      NaN
98           NaN      NaN
99           NaN      NaN

[100 rows x 15 columns]
```

Each row is a single measurement, but it does not say what is being measured. Explain how you could find this out. (Hint: see the 'Data Description' section of the documentation.)

Written answer: Tables are linked by identifiers which have 'ID' as a suffix. For example, SUBJECT_ID refers to a patient, ICUSTAY_ID refers to a specific ICU visit, and HADM_ID refers to a specific hospital visit. Charted events are stored as a series of events tables, such as OUTPUTEVENTS and LABEVENTS. By joining dictionary tables that store identifiers prefixed with "D_" and events tables on ITEMID, we can find out what is being measured. Essentially we are cross-referencing codes stored in dictionary tables against their respective definitions in the events tables.

So far we've looked at the original MIMIC-III database, but it's not in a format suitable for a sequence model prediction (like LSTMs or transformers). You ran two lines of code at the start of the assignment to get it in the right format.

The first function was `parser_utils.do_all_parsing`, which created a set of files in `ROOT/mimic_database/mapped_events`, which are closer to what we need. One output was the file `CHARTEVENTS_reduced_24_hour_blocks_plus_admissions_plus_patients_plus_scripts_plus_icds_plus_no` This file: - Is similar to `CHARTEVENTS`, except that each measurement is **assigned to a single 24hr block** (like 2117-09-11), rather than a specific timestamp (like 2117-09-11 16:04:00). You can think of this as discretizing the dataset. - Each row also has extra data about the patient: admission times, scripts, and known patient diseases (ICD's).

Next you ran `data_utils.load_seq_dataset` which does the following: - Given a prediction target (one of `MI`, `SEPSIS`, or `VANCOMYCIN`), generate numpy arrays for the train, test, and validation set. The data is shuffled before splitting. - Put X-data into the shape `(n_hostpital_stays, n_timesteps, n_features)`. So `X[i,j,k]` gives the kth feature, for the jth day of the ith hospital stay. - Puts the labels, y-data, into shape `(n_hospital_stays,n_timesteps,1)`. All 3 prediction problems are binary, to these values 0 or 1. - Returns a list of strings called `features` containing the names of each feature in the X-data. So `features[k]` is the name of the features in `X[:,:,k]` - Zero-padding. Since not all patients will have a valid measurememt for every feature at each timestep, we fill the remainder with zeros. Later we will tell the model to ignore this data in training. - Z-score normalization.

Now we can load that data using the following function call:

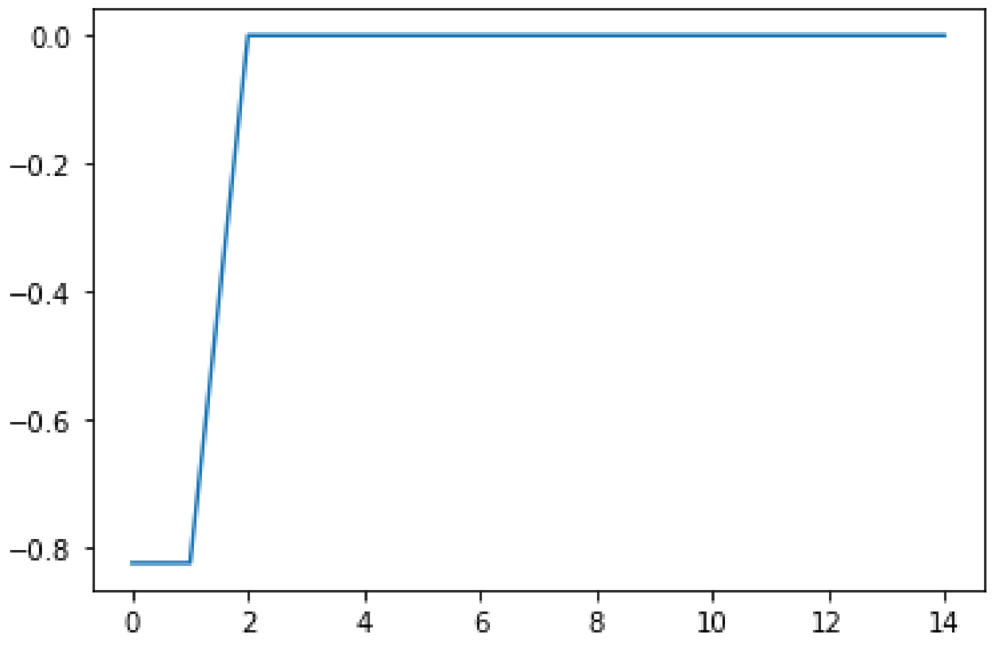
[6]:
```
target='SEPSIS'    # 'SEPSIS' or 'MI' or 'VANCOMYCIN'
train_x, val_x, train_y, val_y, no_feature_cols, test_x, test_y,␣
 ↪x_boolmat_test, y_boolmat_test, x_boolmat_val, y_boolmat_val, features \
    = data_utils.load_seq_dataset(ROOT, target)

print("train shapes ",train_x.shape, train_y.shape)
print("val shapes   ", val_x.shape, val_y.shape)
print("test shapes  ", test_x.shape, test_y.shape)
print("# features   ", len(features))
```

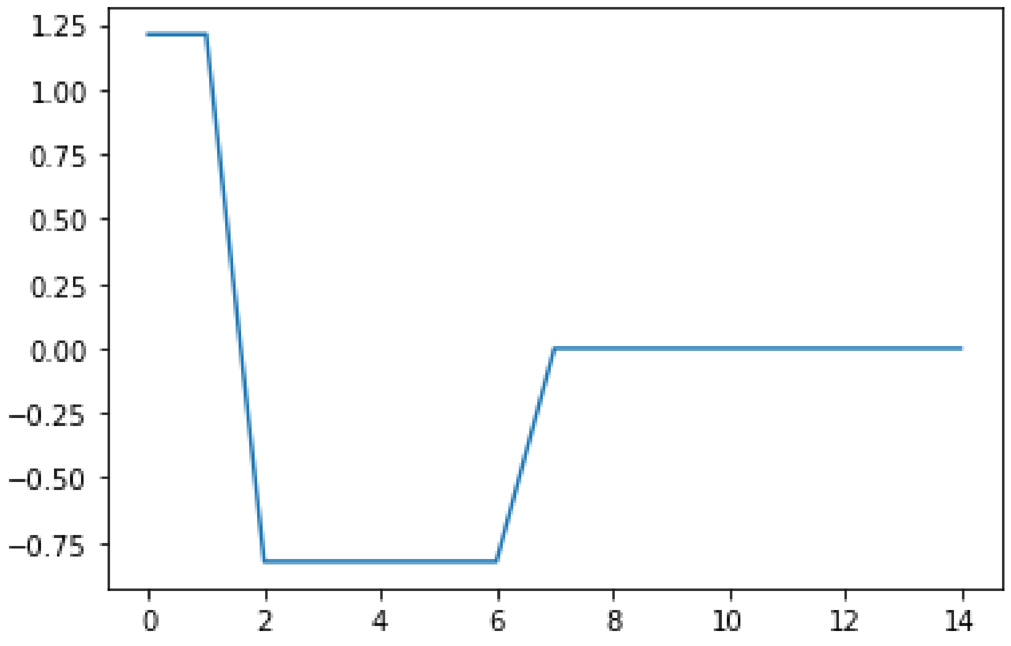
```
train shapes  (2984, 15, 226) (2984, 15, 1)
val shapes    (5178, 15, 226) (5178, 15, 1)
test shapes   (10355, 15, 226) (10355, 15, 1)
# features    226
```

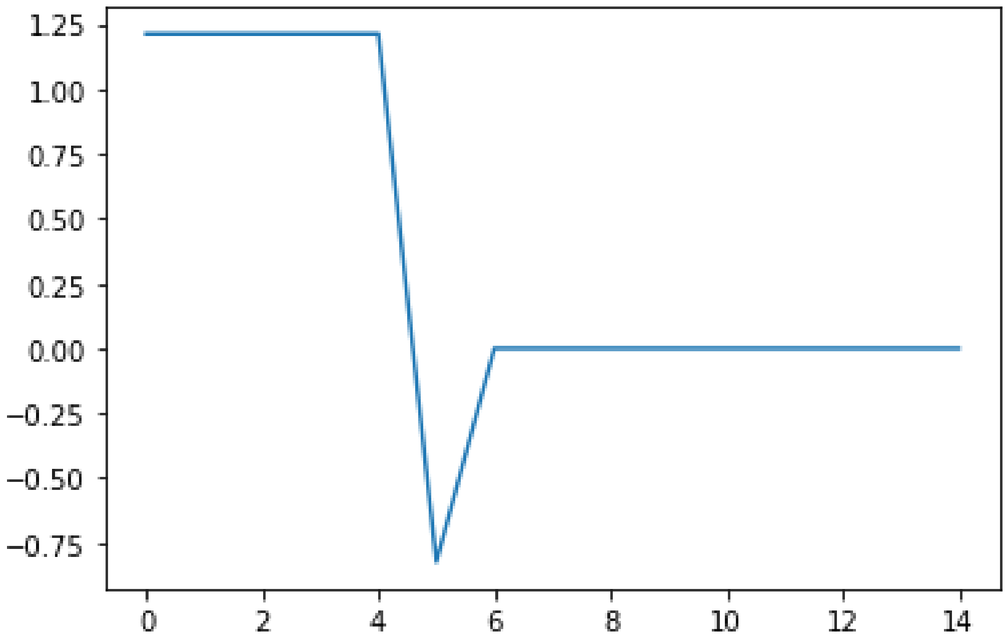
Q1.5 sequence model data sets

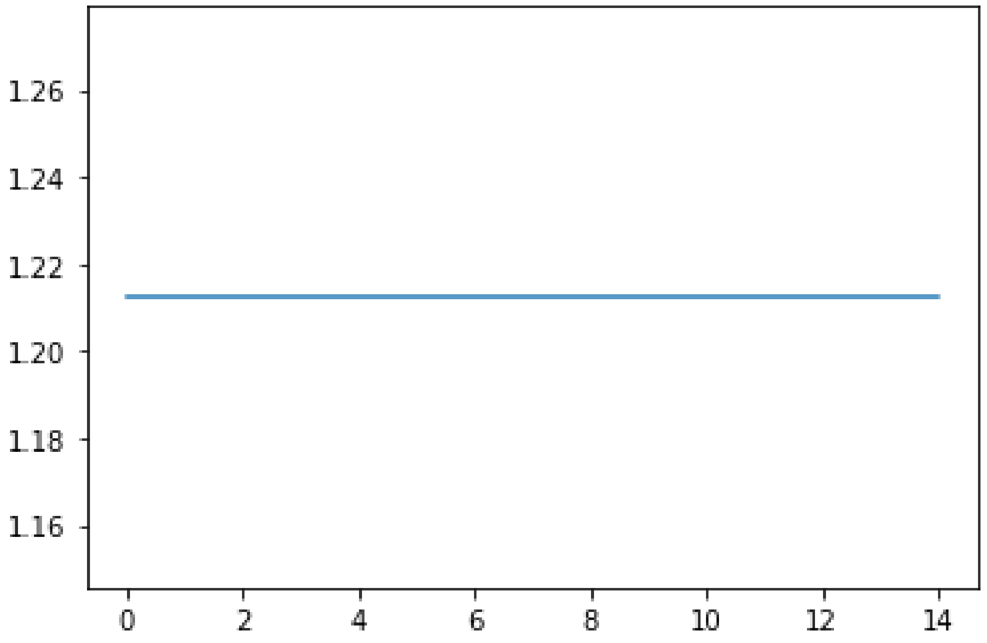
Let's look at some samples from `train_x`. In the below code cell, get the feature called 'HDL', and generate 20 plots showing how this variable changes over all the timesteps for the first 20 hospital stays (1 time series plot per hospital stay).

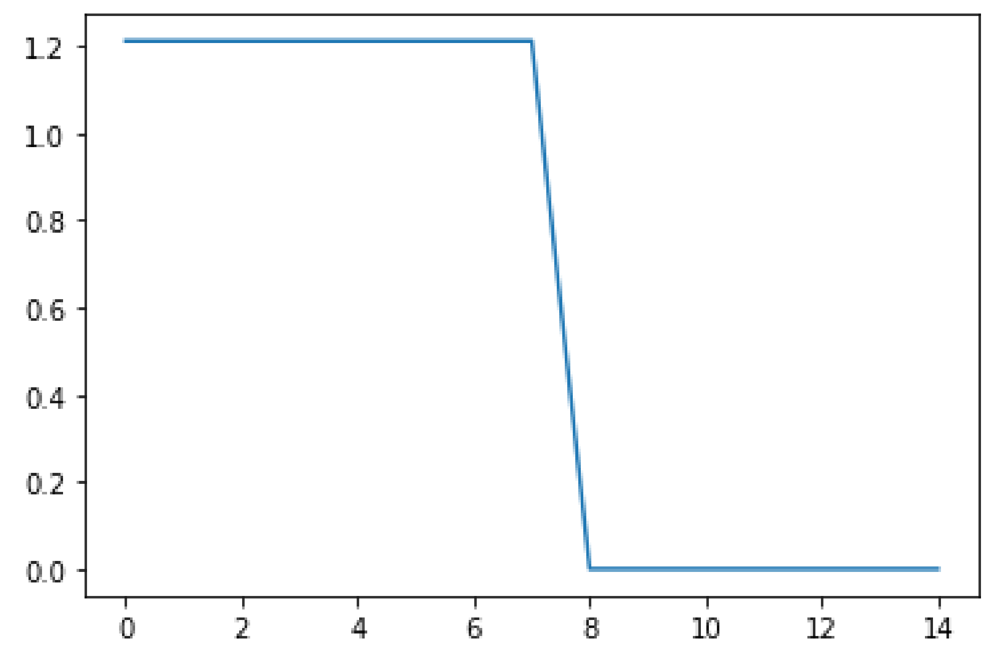
[7]:
```
# YOUR CODE HERE #
num_stays = 20
hdl_i = features.index("HDL")
for i in range(num_stays):
    i_data = train_x[i, :, hdl_i]
    plt.plot(i_data)
    plt.show()


# END CODE #
```



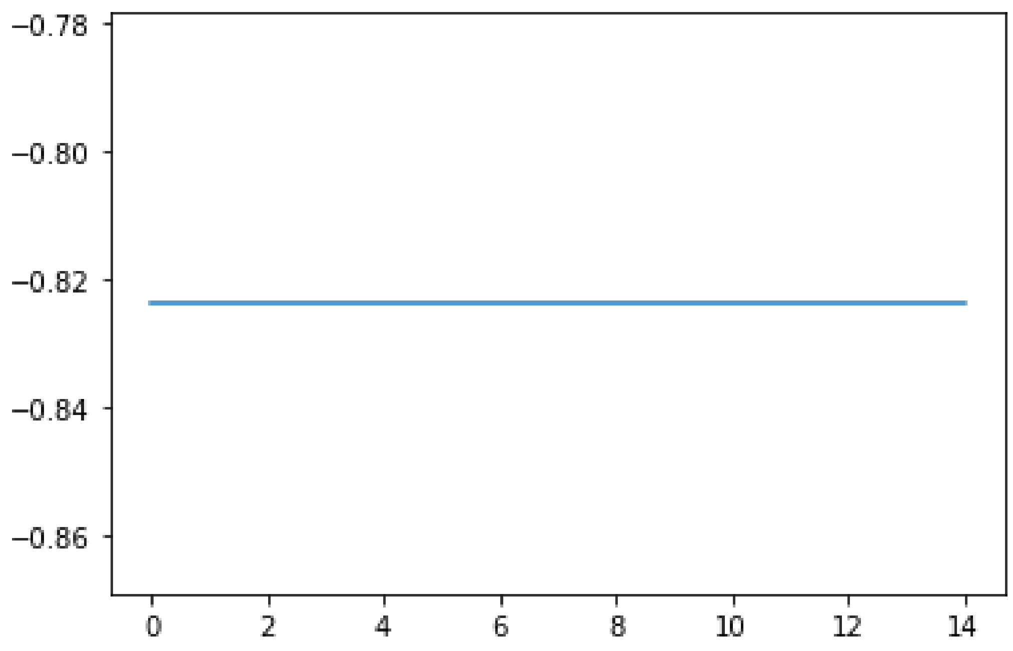



1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
12
14

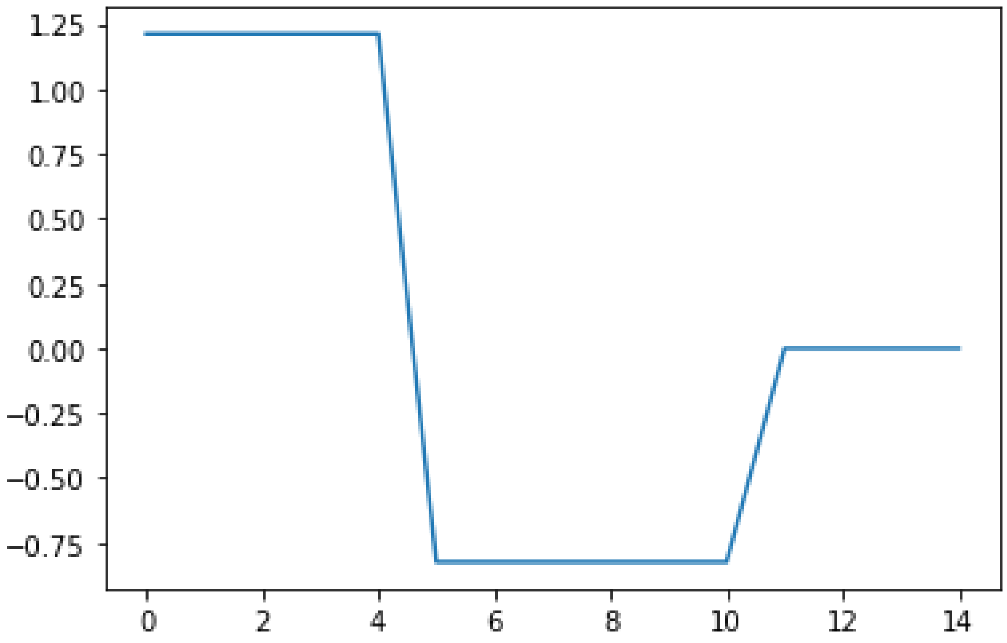

1.26
1.24
1.22
1.20
1.18
1.16
0
2
4
6
8
10
12
14

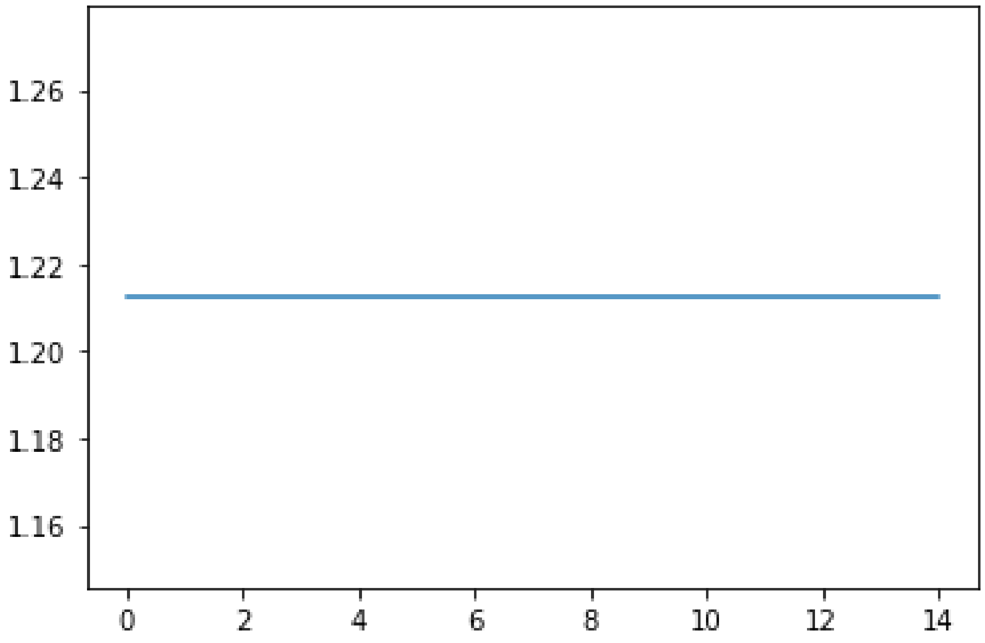


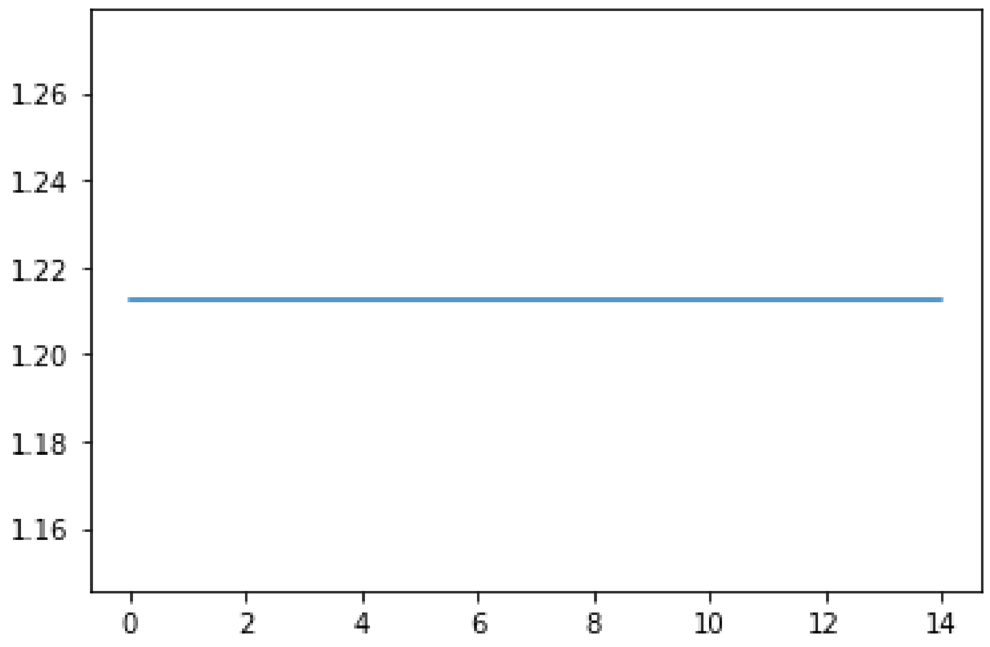
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14

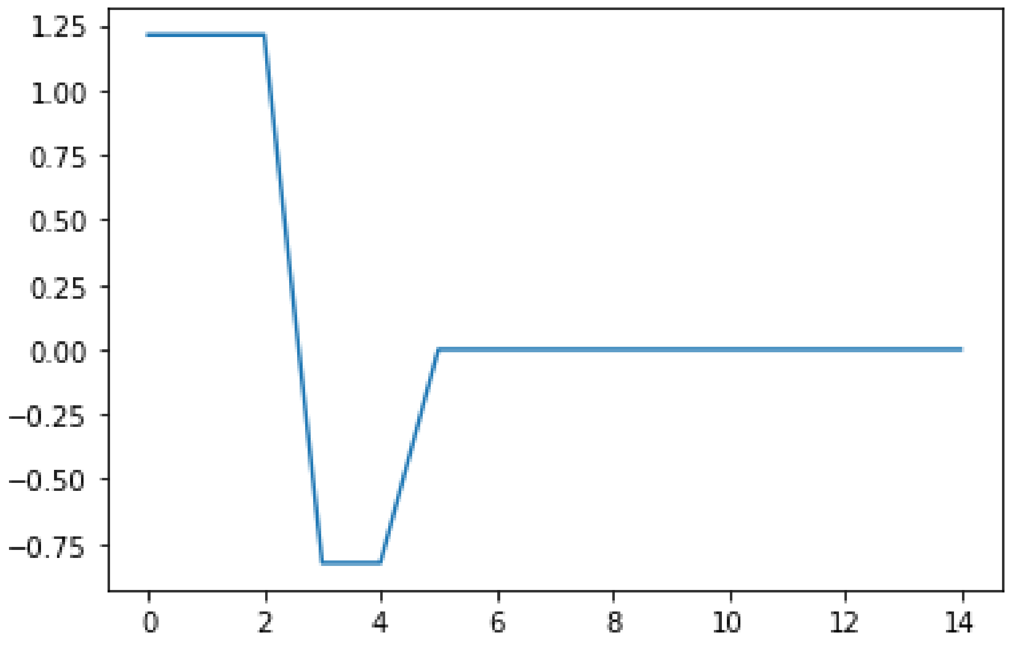


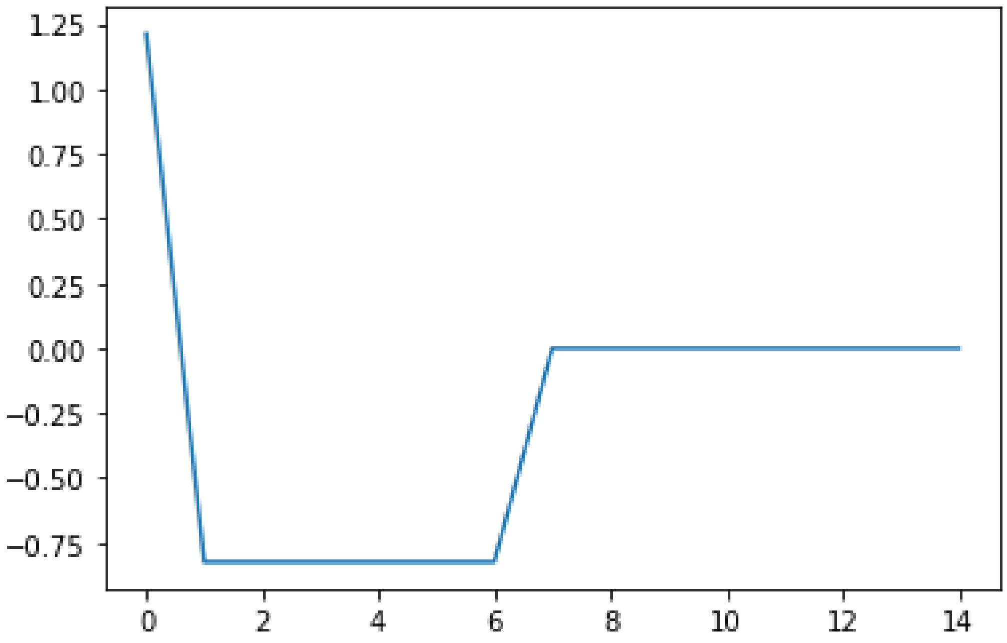
−0.78
−0.80
−0.82
−0.84
−0.86
0
2
4
6
8
10
12
14

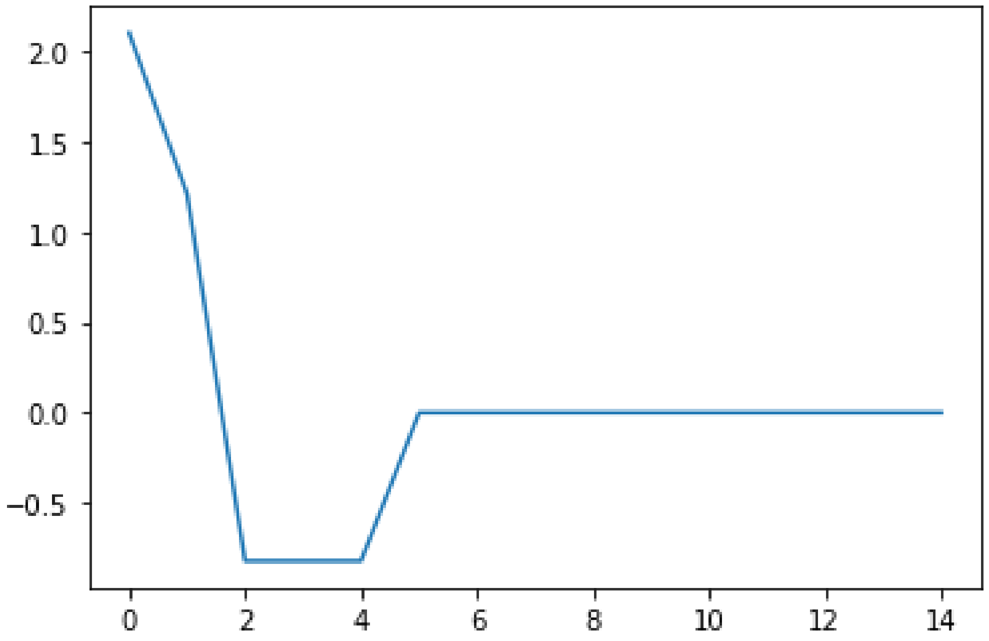

1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
12
14

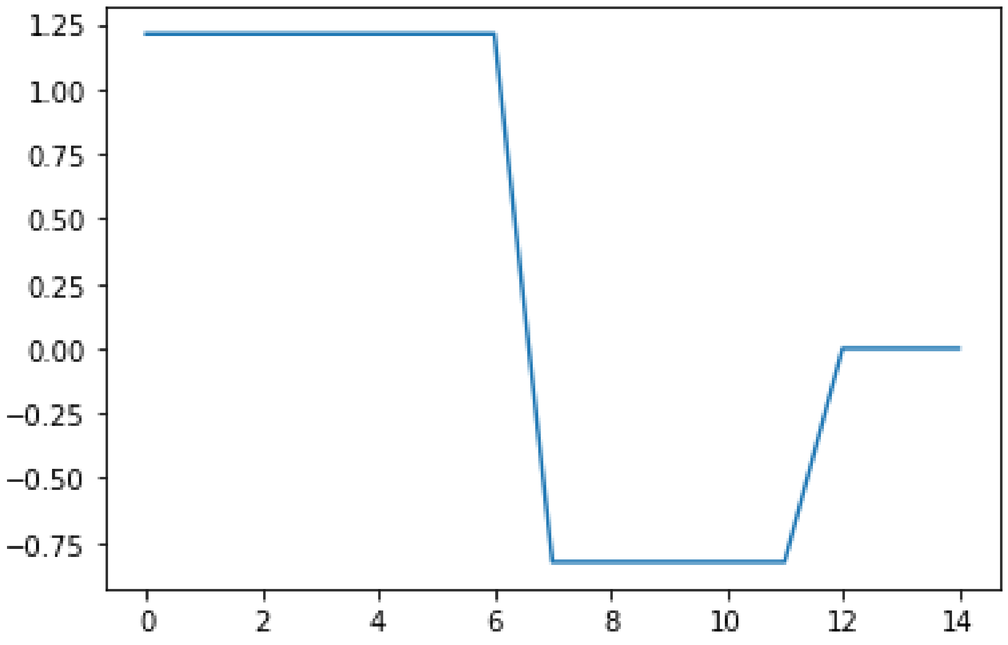

1.26
1.24
1.22
1.20
1.18
1.16
0
2
4
6
8
10
12
14

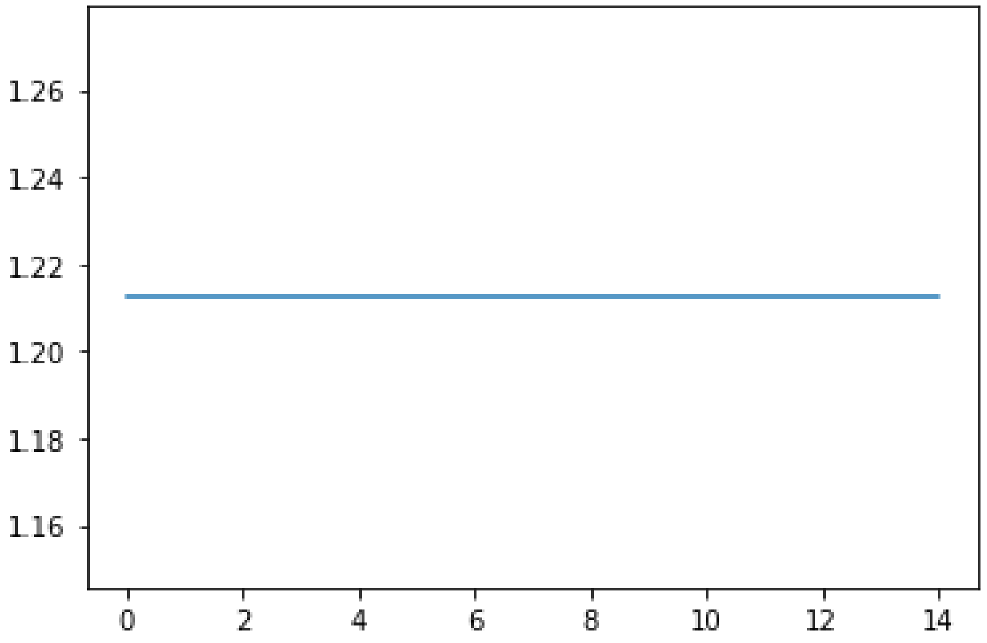

1.26
1.24
1.22
1.20
1.18
1.16
0
2
4
6
8
10
12
14

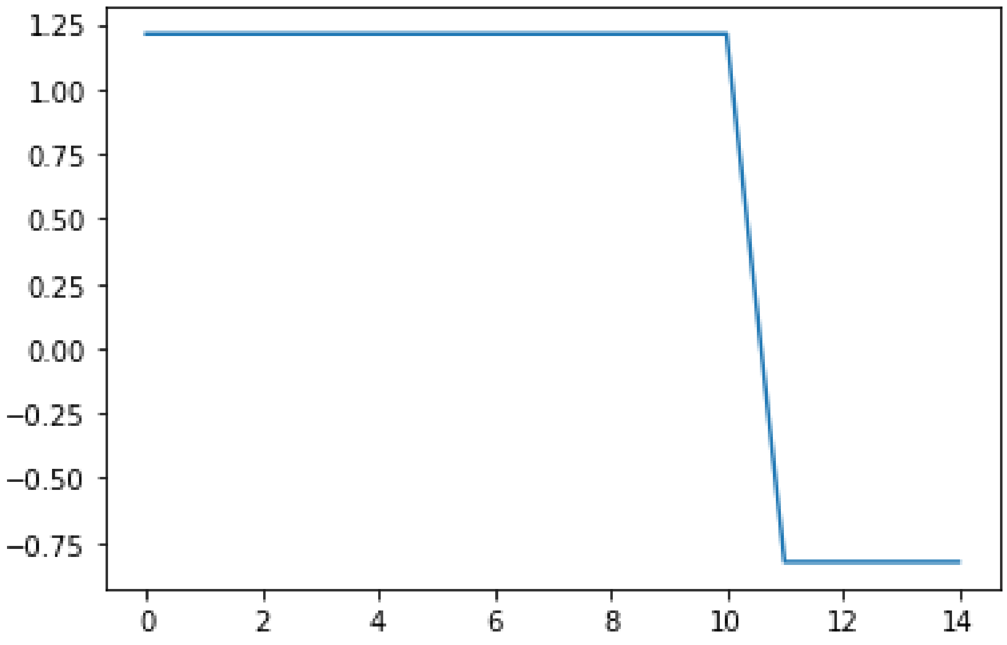

1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
12
14

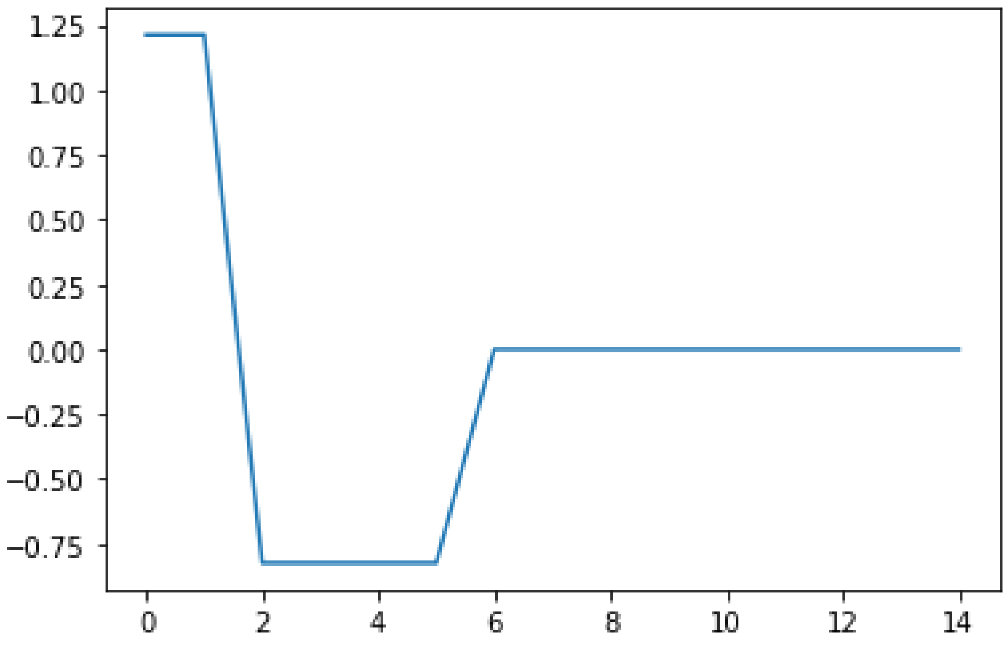

1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
12
14

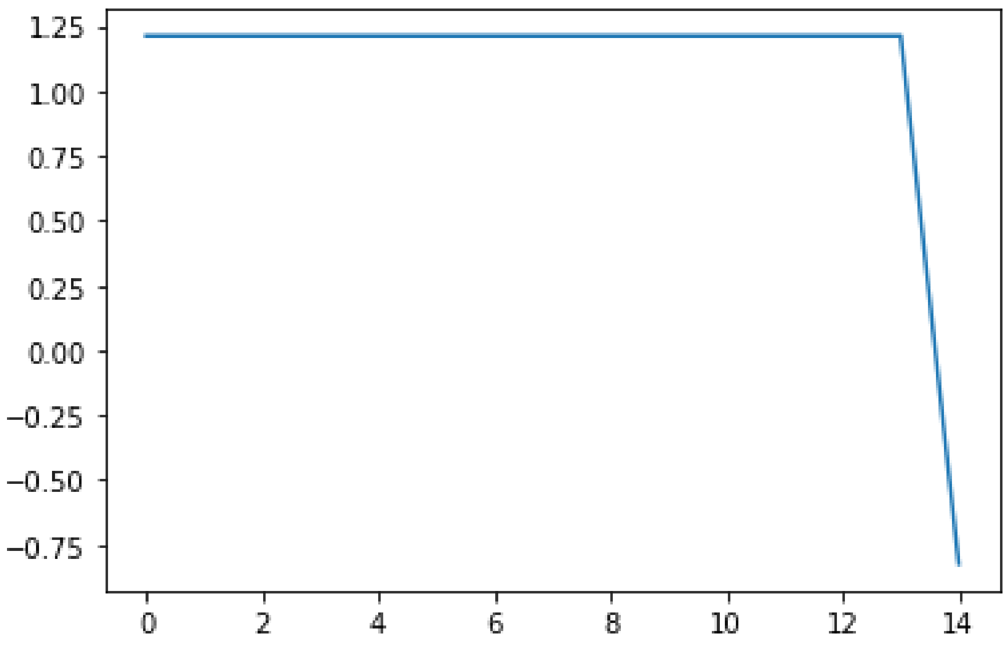

2.0
1.5
1.0
0.5
0.0
−0.5
0
2
4
6
8
10
12
14


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
12
14

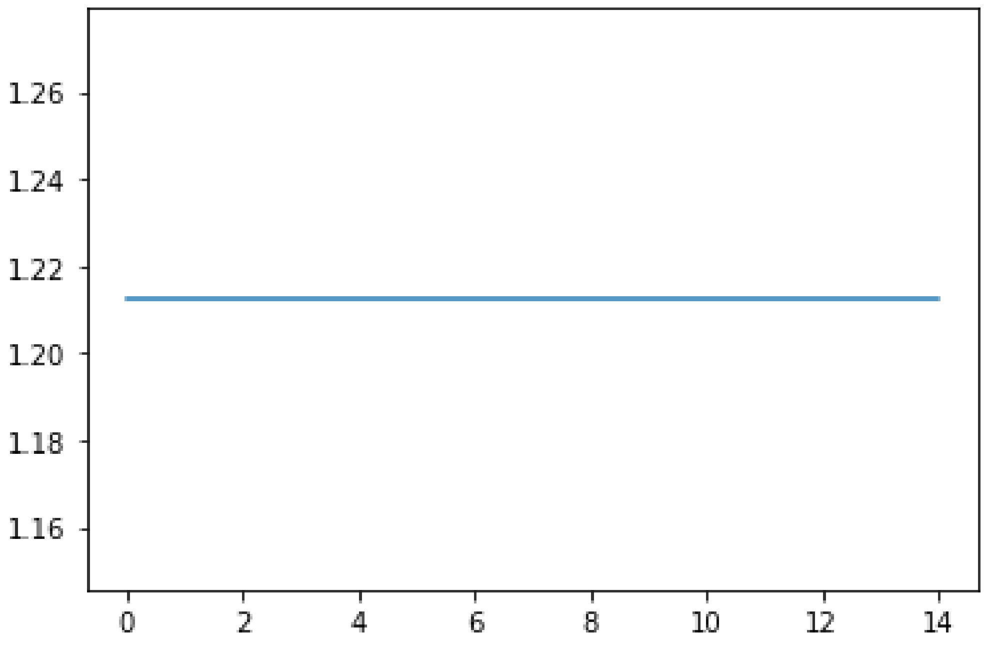

1.26
1.24
1.22
1.20
1.18
1.16
0
2
4
6
8
10
12
14


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
12
14

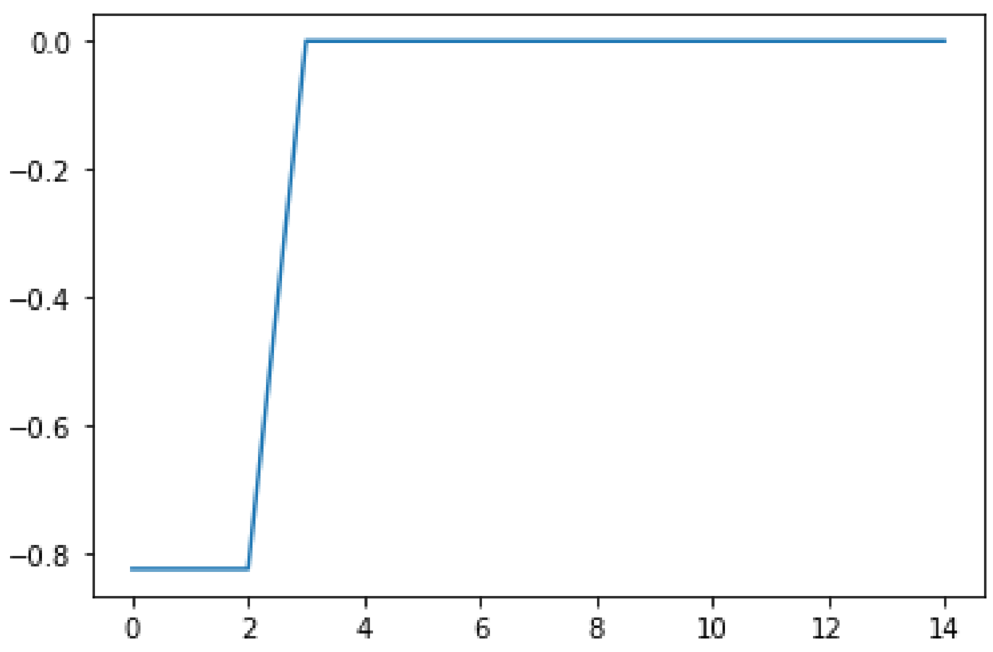

1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
12
14

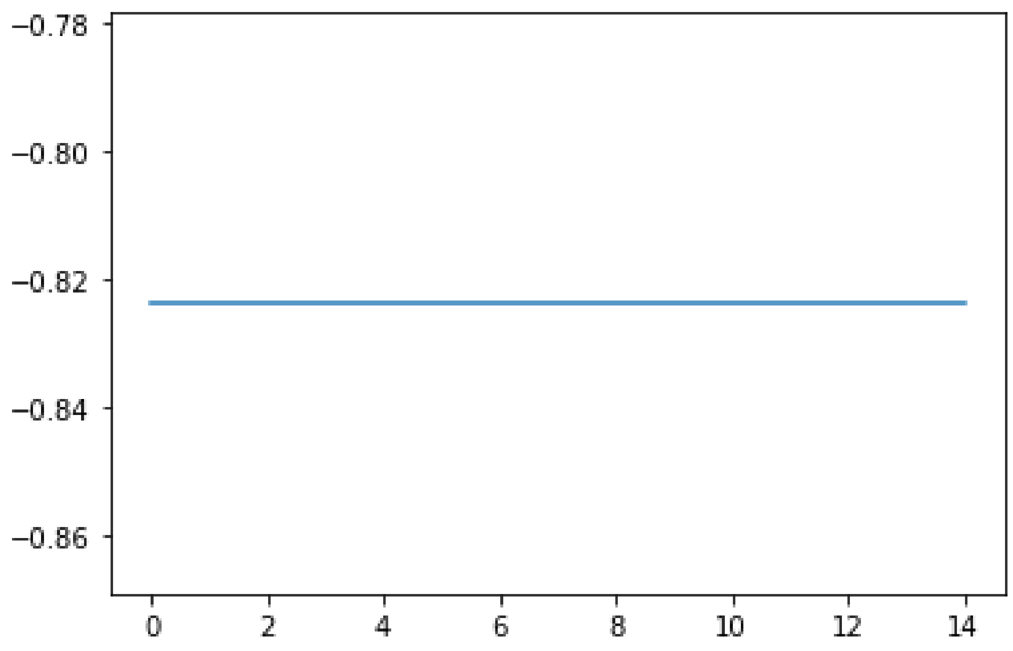


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
12
14



1.26
1.24
1.22
1.20
1.18
1.16
0
2
4
6
8
10
12
14





By looking at these graphs and by carefully reading the description of `return_data` above, answer the following questions: - Why are the last values in the series usually 0, and why do some reach 0 earlier than others? - "HDL" is cholesterol, so how is it possible that some values are actually negative? - In many cases, the measured HDL cholesterol is exactly the same on consecutive days, even though we would expect a measurement to fluctuate at least a bit between days. Why is this the case?

Written answers - . The last values in the series are usually 0, which means that no data has been collected for that timestep. Some tables reach 0 earlier than others, as the patient may have been released from the hospital earlier than others, which cuts out the data available beyond that timestep. In addition, generally the data may not have been entered manually by the attending clinician. - . The HDL is negative as the value is z-score normalized so every value above the mean is positive and below the mean is negative. - . The measured HDL cholesterol values may be exactly the same on consecutive days as a result of clinician human error and inputting the same cholesterol values across the following hours. We need a value for every timestep, so we re-use the same result from the previous hour and as the current hour without measuring it.

Q1.6 prediction problem

In the first cell of this notebook we explained the prediction problem at a high level. Explain the prediction problem again, but be specific in explaining the data inputs and prediction outputs. Using the terminology from lecture, is this a "many-to-many" problem, or a "many-to-one" problem?

Written Answer: By working with structured data measurements such as heart rate, glucose, and central venous pressures as our input values, we'll be making early predictions of sepsis, myocardial infarction, and vancomycin antibiotic administration. As such, this is a many-to-many problem as we are taking multiple structured data measurements as our inputs and outputting one prediction at each timestep of whether the patient is at risk of sepsis, myocardial infarction, and vancomycin antibiotic administration.

1.2 LSTM prediction model

Q1.7 define an LSTM

We'll use an LSTM model to predict a patient outcome at each time step. So each data point we pass to the model will be a sequence of length `n_timesteps`, where each timestamp has `k` features. Our prediction output is also a sequence of length `n_timesteps`.

Using Keras `Sequential`, define a model called `model_lstm`. It should have: - 1 LSTM layer with 256 units. Use the default activation for the LSTM layer. Keras has optimized GPU implementations for most layers, but it does not have an optimized implementation for LSTM with non-default activations. - 1 dropout layer with `rate=0.5`. - 1 dense layer that applies the same tranformation to each of the prediction outputs. - A suitable activation function for the prediction task.

[8]:
```python
def build_lstm_model(lstm_hidden_units= 256):
    """
    Return a simple Keras model with a single LSTM layer, dropout later,
    and then dense prediction layer.

    Args:
    lstm_hidden_units (int): units in the LSTM layer

    Returns:
    model_lstm (tf.keras.Model) LSTM keras model with output dimension (None,1)
    """
    model_lstm = None
    # YOUR CODE HERE #
    model_lstm = tf.keras.Sequential(
        layers=[
            tf.keras.layers.LSTM(lstm_hidden_units, return_sequences=True),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(1, activation='sigmoid')
        ]
    )
    # END CODE #
    return model_lstm

# test code for checking the shape #
lstm_hidden_units = 256
model_lstm = build_lstm_model(lstm_hidden_units)
bs=8
x_batch = train_x[:bs]
print(model_lstm(x_batch).shape)  # expect shape (8, 15, 1) # batch size 8, 15␣
 ↪timesteps
```

```
2021-10-30 02:05:01.971847: 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
2021-10-30 02:05:01.982831: 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
2021-10-30 02:05:01.983596: 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
2021-10-30 02:05:01.986266: 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 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-10-30 02:05:01.986719: 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
2021-10-30 02:05:01.987741: 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
2021-10-30 02:05:01.988505: 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
2021-10-30 02:05:02.395285: 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
2021-10-30 02:05:02.396099: 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
2021-10-30 02:05:02.396857: 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
2021-10-30 02:05:02.397583: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 10819 MB memory:  -> device:
0, name: Tesla K80, pci bus id: 0000:00:04.0, compute capability: 3.7
2021-10-30 02:05:03.027770: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369]
Loaded cuDNN version 8005
```

```
(8, 15, 1)
```

Q1.8 masking

Read about the masking layer in Keras. Briefly explain why we need masking for this problem. Your answer should refer back to the time-series plots generated in Q1.5.

Written answer: The masking layer is used to mask a sequence by using a mask value to skip timesteps. We need masking for this problem, because as shown in the time-series plots generated in Q1.5, we have missing timesteps in our input, which we need to skip when processing the data as that would affect our final model prediction accuracy and efficiency.

The below function builds the final model. We provide code that calls `build_lstm_model`.

Your code should add a masking layer with `mask_value=0`, and it should be applied at the start of the model.

```
[9]: def build_masked_lstm_model(num_timesteps, num_features, lstm_hidden_units=256):
         """
         Return a simple Keras model with a masking single LSTM layer, dropout␣
     ↪later,
         and then dense prediction layer.

         Args:
         num_timesteps (int): num timesteps per input data object.
         num_features (int): num features per input data object.
         lstm_hidden_units (int): units in the LSTM layer
```

```
    Returns:
    model_lstm (tf.keras.Model) LSTM keras model with output dimension (None,1)
    """
    model_lstm = build_lstm_model(lstm_hidden_units)
    for layer in model_lstm.layers:
        layer.supports_masking=True

    model = None
    # YOUR CODE HERE #
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Masking(mask_value=0.,
                                  input_shape=(num_timesteps, num_features)))
    model.add(model_lstm)
    # END CODE #
    return model

# Code to test the shape is correc #
num_timesteps, num_features = train_x.shape[-2:]
lstm_hidden_units = 256

model = build_masked_lstm_model(num_timesteps, num_features, lstm_hidden_units)
bs=8
x_batch = train_x[:bs]
print(model(x_batch).shape)  # expect shape (8, 15, 1) # batch size 8, 15␣
 ↪timesteps
```

```
(8, 15, 1)
```

Q1.9 compiling and training

Below we have copied the code for getting the dataset that we ran earlier. You can choose 'MI'', SEPSIS' or 'VANCOMYCIN' as the target. When submitting this assignment, please choose 'VANCOMYCIN'.

The code also calls `build_masked_lstm_model` that you just defined. Note that the model parameters depend on the dataset shape, so if we wanted to change the dataset from 'MI' to 'SEPSIS' then we need to create the model again with different input shapes. We chose to define the model inside a function so that we could re-create the model more easily.

Compile the model using - The Adam otptimizer with default parameters. - An appropriate loss function for this task. - Metrics: accuracy and tensorflow's AUC

[10]:
```
target='VANCOMYCIN'   # 'SEPSIS' or 'MI' or 'VANCOMYCIN'
train_x, val_x, train_y, val_y, no_feature_cols, test_x, test_y,␣
 ↪x_boolmat_test, y_boolmat_test, x_boolmat_val, y_boolmat_val, features \
    = data_utils.load_seq_dataset(ROOT, target)
num_timesteps, num_features = train_x.shape[-2:]
model = build_masked_lstm_model(num_timesteps, num_features, lstm_hidden_units)
```

```
# YOUR CODE HERE #
model.compile(loss=tf.keras.losses.BinaryCrossentropy(), optimizer=tf.keras.
↪optimizers.Adam(learning_rate=0.001), metrics=['accuracy', tf.keras.metrics.
↪AUC()])
# YOUR CODE HERE #
```

Finally fit the model. It should train very quickly. For 'VANCOMYCIN', validation accuracy should be around 0.85. For 'MI' it should be above 0.95.

```
[11]: epochs=10
# YOUR CODE HERE #
hist = model.fit(x=train_x, y=train_y, epochs=epochs, validation_data=(val_x,␣
↪val_y))
# YOUR CODE HERE #
```

```
2021-10-30 02:05:07.346604: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)

Epoch 1/10
914/914 [==============================] - 22s 19ms/step - loss: 0.2413 -
accuracy: 0.8110 - auc: 0.8698 - val_loss: 0.1933 - val_accuracy: 0.8413 -
val_auc: 0.9005
Epoch 2/10
914/914 [==============================] - 16s 17ms/step - loss: 0.2143 -
accuracy: 0.8337 - auc: 0.8995 - val_loss: 0.1860 - val_accuracy: 0.8484 -
val_auc: 0.9088
Epoch 3/10
914/914 [==============================] - 16s 17ms/step - loss: 0.2055 -
accuracy: 0.8413 - auc: 0.9081 - val_loss: 0.1802 - val_accuracy: 0.8541 -
val_auc: 0.9149
Epoch 4/10
914/914 [==============================] - 16s 17ms/step - loss: 0.1986 -
accuracy: 0.8463 - auc: 0.9146 - val_loss: 0.1828 - val_accuracy: 0.8528 -
val_auc: 0.9119
Epoch 5/10
914/914 [==============================] - 16s 17ms/step - loss: 0.1924 -
accuracy: 0.8521 - auc: 0.9202 - val_loss: 0.1797 - val_accuracy: 0.8544 -
val_auc: 0.9161
Epoch 6/10
914/914 [==============================] - 16s 17ms/step - loss: 0.1861 -
accuracy: 0.8576 - auc: 0.9257 - val_loss: 0.1784 - val_accuracy: 0.8552 -
val_auc: 0.9172
Epoch 7/10
914/914 [==============================] - 16s 17ms/step - loss: 0.1811 -
accuracy: 0.8612 - auc: 0.9298 - val_loss: 0.1801 - val_accuracy: 0.8542 -
val_auc: 0.9165
Epoch 8/10
```

```
914/914 [==============================] - 16s 17ms/step - loss: 0.1745 -
accuracy: 0.8665 - auc: 0.9350 - val_loss: 0.1820 - val_accuracy: 0.8526 -
val_auc: 0.9148
Epoch 9/10
914/914 [==============================] - 16s 17ms/step - loss: 0.1681 -
accuracy: 0.8735 - auc: 0.9401 - val_loss: 0.1850 - val_accuracy: 0.8531 -
val_auc: 0.9134
Epoch 10/10
914/914 [==============================] - 16s 17ms/step - loss: 0.1588 -
accuracy: 0.8817 - auc: 0.9466 - val_loss: 0.1931 - val_accuracy: 0.8503 -
val_auc: 0.9108
```

1.2.1 Evaluation

Q1.10 predition and masking

Use `model.predict()` on the test dataset and save predictions to the variable `test_y_pred`.

[12]:
```python
test_y_pred = None
# YOUR CODE HERE #
test_y_pred = model.predict(test_x)
# END CODE #

print(test_y_pred.shape) # expect (n_datapoints, 15, 1)
```

```
(10355, 15, 1)
```

Normally when we we compare a set of predictions, we'd take 2 vectors: `y_true` which is a flat vector of 0s and 1s, and `y_pred` which is a vector of the same shape with probabilities in the range [0,1]. Our case is different because: - Our prediction output has an extra axis `(None,timesteps,features)` instead of `(None,features)`. - Some of the predictions should be masked, and therefore removed from the final prediction dataset.

The earlier function `data_utils.load_seq_dataset` returned a mask vector `y_boolmat_test` with the same shape as `test_y`. If `y_boolmat_test[i]==True` then this label should be masked (removed from the evaluation dataset).

In the next cell use `test_y_pred` and `y_boolmat_test` to create the vectors `y_pred_masked` and `y_true_masked` by removing the masked predictions, and flattening the output.

[13]:
```python
y_pred_masked = None
y_true_masked = None

# YOUR CODE HERE #
y_pred_masked = []
y_true_masked = []
for i in range(y_boolmat_test.shape[0]):
    for n in range(y_boolmat_test.shape[1]):
        if y_boolmat_test[i, n, 0] == False:
            y_pred_masked.append(test_y_pred[i, n, 0])
            y_true_masked.append(test_y[i, n, 0])
y_pred_masked = np.array(y_pred_masked)
y_true_masked = np.array(y_true_masked)
# END CODE #
print(y_pred_masked.shape, y_true_masked.shape) # expect shape (n_predictions,)␣
 ↪and the shape should be the same
```

```
(87373,) (87373,)
```

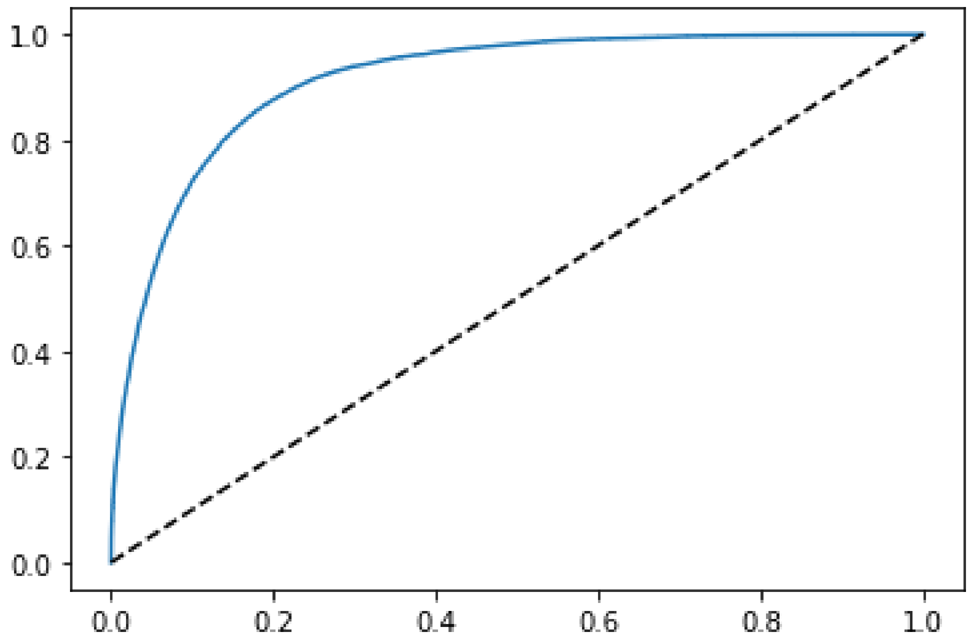
Q1.11 ROC+AUC

Using `y_pred_masked` and `y_true_masked`: - Plot a ROC curve. - Print the AUC.

You can use the functions in `sklearn.metrics` for both.

[14]:
```
# YOUR CODE HERE #
from sklearn.metrics import roc_curve, roc_auc_score
fpr, tpr, thresholds = roc_curve(y_true_masked, y_pred_masked)
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], 'k--')
auc = roc_auc_score(y_true_masked, y_pred_masked)
print(f"AUC score={auc}")
# END CODE #
```

```
AUC score=0.9130357116229655
```



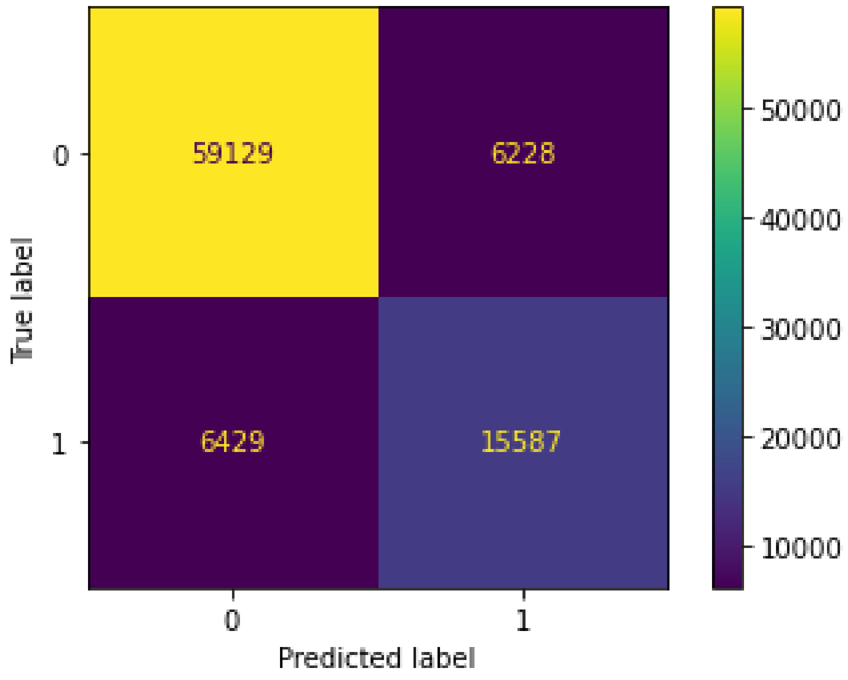
Q1.12 confusion

Finally, generate a confusion matrix. To do this you will need to convert prediction probabilities in `y_pred_masked` to binary predictions. You can choose a threshold of 0.5. Again, you can use functions from `sklearn.metrics`.

You can use a plotting library to display the confusion matrix, but you can also just print the array directly. If you do just print the array, then also print a message explaining the each axis.

[15]:
```
# YOUR CODE HERE #
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import binarize
threshold = 0.5
for i in range(y_pred_masked.shape[0]):
    if y_pred_masked[i] < threshold:
        y_pred_masked[i] = 0
    else:
        y_pred_masked[i] = 1
confusion = confusion_matrix(y_true_masked, y_pred_masked)
disp = ConfusionMatrixDisplay(confusion)
disp.plot()
# END CODE #
```

[15]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f9b63fa84d0>
```



Q1.13 clinical application tradeoffs

We have focused on modelling sepsis, but now consider vancomycin prediction. Explain what it means to choose different operating points at different positions in the ROC curve. Specifically tie it back to the use case of vancomycin antibiotic adminsitration. What are the tradeoffs? Pick a two points on the ROC curve and explain what the true positive and false positive rates mean at those points.

Written answer: To choose different operating points at different positions in the ROC curve means to choose tradeoffs between specificity and sensitivity. An operating point with high sensitivity corresponds to high negative predictive value, which is ideal for "rule-out" tests. On the other hand, an operating point with high specificity corresponds to high positive predictive value, which is ideal for "rule-in" tests. For high specificity, it can be used to test negativity in health, where we want to know the proportion of population without the disease and give negative test results. High specificity should be used to make decisions about high-risk actions. For our specific use case of vancomycin antibiotic administration, it is important to select an effective operating point as a guideline for dosing, as it's necessary to balance nephrotoxicity with vancomycin's antibiotic activity. In addition, it's important to not over-dose antibiotics to reduce incidence of antibiotic resistance. The most optimal operating point to choose is usually when the classifier gives the best trade off between the costs of failing to detect positives against the costs of raising false alarms. The first point on the ROC curve I'll choose is when both the FPR and TPR are low, when the fpr is equal to 0.05. At this point, the TPR is 0.6, which is a relatively strict threshold to choose. At this point, the true positive rate means how many correct positive results occur among all positive samples available, which is 0.6, while the fpr is 0.05, which defines how many incorrect positive results occur among all negative samples available during the test. The second point on the ROC curve I'll choose is when FPR is 0.6 and TPR is around 0.95. At this point, the model predicts 0.95 correct positive results among all positive. samples and 0.6 incorrect positive results among all negative samples. At these two extremes, we can see the clear tradeoffs between sensitivity vs. specificity.

```
[ ]:
```

A2_part2_clinical_Word2Vec_embeddings_and_readmission_prediction

October 30, 2021

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.

Written answer: A model for 30-day readmission prediction can be used to reduce readmission risk by assessing doctor notes and predicting whether a patient is readmitted. By predicting whether a patient is readmitted, clinicians can focus on complementing inpatient care with postdischarge interventions and/or enhanced care transition. Early interventions during inpatient hospitalization such as early discharge planning can serve to reduce readmissions. As health care resources are limited, it's essential to predict which patients are at highest risk of readmission and invest readmission-preventative interventions to these specific patients to reduce 30-day readmissions.

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 https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.2.5/
↪en_core_sci_md-0.2.5.tar.gz
```

```
Requirement already satisfied: gensim in /opt/conda/lib/python3.7/site-packages
(4.1.2)
Requirement already satisfied: smart-open>=1.8.1 in
/opt/conda/lib/python3.7/site-packages (from gensim) (5.2.1)
Requirement already satisfied: scipy>=0.18.1 in /opt/conda/lib/python3.7/site-
packages (from gensim) (1.7.1)
Requirement already satisfied: numpy>=1.17.0 in /opt/conda/lib/python3.7/site-
packages (from gensim) (1.19.5)
Requirement already satisfied: spacy==2.3.7 in /opt/conda/lib/python3.7/site-
packages (2.3.7)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (1.0.5)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (0.8.2)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (1.0.0)
Requirement already satisfied: thinc<7.5.0,>=7.4.1 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (7.4.5)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (4.62.3)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (1.0.5)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (2.0.5)
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: requests<3.0.0,>=2.13.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (2.25.1)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (0.7.4)
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: setuptools in /opt/conda/lib/python3.7/site-
packages (from spacy==2.3.7) (58.0.4)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/opt/conda/lib/python3.7/site-packages (from spacy==2.3.7) (3.0.5)
Requirement already satisfied: importlib-metadata>=0.20 in
/opt/conda/lib/python3.7/site-packages (from
catalogue<1.1.0,>=0.0.7->spacy==2.3.7) (4.8.1)
Requirement already satisfied: typing-extensions>=3.6.4 in
/opt/conda/lib/python3.7/site-packages (from importlib-
metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy==2.3.7) (3.10.0.2)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from importlib-metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy==2.3.7)
```

```
(3.5.0)
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) (2021.5.30)
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) (1.26.6)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/site-
packages (from requests<3.0.0,>=2.13.0->spacy==2.3.7) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0,>=2.13.0->spacy==2.3.7) (4.0.0)
Requirement already satisfied: scispacy==0.3.0 in /opt/conda/lib/python3.7/site-
packages (0.3.0)
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: 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: numpy in /opt/conda/lib/python3.7/site-packages
(from scispacy==0.3.0) (1.19.5)
Requirement already satisfied: requests<3.0.0conllu,>=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from scispacy==0.3.0) (2.25.1)
Requirement already satisfied: scikit-learn>=0.20.3 in
/opt/conda/lib/python3.7/site-packages (from scispacy==0.3.0) (0.24.2)
Requirement already satisfied: joblib in /opt/conda/lib/python3.7/site-packages
(from scispacy==0.3.0) (1.0.1)
Requirement already satisfied: pysbd in /opt/conda/lib/python3.7/site-packages
(from scispacy==0.3.0) (0.3.4)
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: psutil in /opt/conda/lib/python3.7/site-packages
(from nmslib>=1.7.3.6->scispacy==0.3.0) (5.8.0)
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) (2021.5.30)
Requirement already satisfied: chardet<5,>=3.0.2 in
/opt/conda/lib/python3.7/site-packages (from
requests<3.0.0conllu,>=2.0.0->scispacy==0.3.0) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/site-
packages (from requests<3.0.0conllu,>=2.0.0->scispacy==0.3.0) (2.10)
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) (1.26.6)
Requirement already satisfied: scipy>=0.19.1 in /opt/conda/lib/python3.7/site-
packages (from scikit-learn>=0.20.3->scispacy==0.3.0) (1.7.1)
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) (2.2.0)
```

```
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) (1.0.5)
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) (1.0.0)
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) (1.0.5)
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) (3.0.5)
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) (0.8.2)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.7/site-
packages (from spacy<3.0.0,>=2.3.0->scispacy==0.3.0) (58.0.4)
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: 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) (2.0.5)
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) (0.7.4)
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) (7.4.5)
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) (4.62.3)
Requirement already satisfied: importlib-metadata>=0.20 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) (4.8.1)
Requirement already satisfied: typing-extensions>=3.6.4 in
/opt/conda/lib/python3.7/site-packages (from importlib-
metadata>=0.20->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: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from importlib-
metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy<3.0.0,>=2.3.0->scispacy==0.3.0)
(3.5.0)
Requirement already satisfied: nltk in /opt/conda/lib/python3.7/site-packages
(3.6.5)
Requirement already satisfied: joblib in /opt/conda/lib/python3.7/site-packages
(from nltk) (1.0.1)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages
```

```
(from nltk) (4.62.3)
Requirement already satisfied: regex>=2021.8.3 in /opt/conda/lib/python3.7/site-
packages (from nltk) (2021.8.28)
Requirement already satisfied: click in /opt/conda/lib/python3.7/site-packages
(from nltk) (8.0.1)
Requirement already satisfied: importlib-metadata in
/opt/conda/lib/python3.7/site-packages (from click->nltk) (4.8.1)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from importlib-metadata->click->nltk) (3.5.0)
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: 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) (4.62.3)
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)
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: 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) (4.62.3)
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) (0.7.4)
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) (2.25.1)
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) (1.0.0)
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) (7.4.5)
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) (2.0.5)
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) (1.0.5)
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) (0.8.2)
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) (1.0.5)
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) (3.0.5)
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: 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) (58.0.4)
Requirement already satisfied: importlib-metadata>=0.20 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) (4.8.1)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from importlib-
metadata>=0.20->catalogue<1.1.0,>=0.0.7->spacy>=2.3.0->en-core-sci-md==0.2.5)
(3.5.0)
Requirement already satisfied: typing-extensions>=3.6.4 in
/opt/conda/lib/python3.7/site-packages (from importlib-
metadata>=0.20->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: 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) (1.26.6)
Requirement already satisfied: idna<3,>=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)
(2.10)
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) (2021.5.30)
Requirement already satisfied: chardet<5,>=3.0.2 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) (4.0.0)
```

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/marchuo/assign2"    # 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/marchuo/assign2/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 #
admissions_df = pd.read_csv(ROOT + "/mimic_database/ADMISSIONS.csv")
display(admissions_df)

insurance_df = admissions_df.groupby(["INSURANCE"])
print(insurance_df.size())
# 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]
```

```
INSURANCE
Government     1783
Medicaid       5785
Medicare      28215
Private       22582
Self Pay        611
dtype: int64
```

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]:
```
# YOUR CODE HERE #
first_nrows = 10
note_df = pd.read_csv(ROOT + "/mimic_database/NOTEEVENTS.csv",␣
 ↪nrows=first_nrows)
print(note_df)
print(note_df["TEXT"].iloc[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 #
print("Label 0 count: " + str(labels_admission.count(0)))
print("Label 1 count: " + str(labels_admission.count(1)))
# YOUR CODE HERE #
```

```
Label 0 count: 502
Label 1 count: 449
```

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]:
```python
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`.

Written answer: notes_seq transforms each note in notes to a sequence of integers. This list of integer sequences encodes the words in our notes, which means the words are replaced by its corresponding integer value from the word_index dictionary, which is the 500 most common words.

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, vocab_words_counts = list(word_counts.keys()),␣
 ↪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 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?

Written Answer: One of the main shortcomings of the skip-gram model for learning word embeddings is that words that are more than 2 words away from the target word are not considered.

As such, some sentences that have important context words that are more than 2 words away are instead not considered and thus will affect the accuracy of our skip-gram model. In addition, skip-gram models fail to consider combined word phrases and the nuance of polysemy - for example "New York" is a single word, but the skip-gram model treats it as two separate words "New" "York".

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 #
        for i in range(len(note)):
            target = note[i]
            slc1 = slice( max(0,i-context_window), i)
            slc2 = slice( i+1, i+context_window+1)

            context = note[slc1] + note[slc2]
            if len(context) != 2*context_window:
                continue
            targets.append(target)
            positive_contexts.append(np.array(context))
        # 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, 244.28it/s]
```

```
(1542542,) (1542542, 4)

Start of the dataset:
[50, 61, 1, 29, 61, 1, 61, 5, 323, 1, 320, 358, 120, 351, 165, 1, 332, 68, 319,
266]

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               [  5 323 320 358]
320             [323   1 358 120]
358             [  1 320 120 351]
```

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]:
```
### 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 (64398,), (64398, 4)
```

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

Written answer: By subsampling frequent words, we are able to obtain significant speedup and also learn more regular word representations. As a result, we improve accuracy of representation of less frequent words and better vector representations of frequent words. As a result, we counter the imbalance between rare and infrequent 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]

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

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

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=20` 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,ing]):  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 #
        t = targets_subsampled[i]
        postive_context = positive_contexts_subsampled[i]
        neg_samples = get_negative_samples(t, postive_context, n_p*num_ns,␣
↪vocab_size)
        negative_contexts_subsampled[i] = neg_samples

    # 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%|                  | 64398/64398 [00:12<00:00, 5324.25it/s]

(64398,) (64398, 4) (64398, 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)
```

```
(257592, 1) (257592, 1) (257592, 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
```

```python
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])
```

```
<BatchDataset shapes: (((1024, 1), (1024, 5)), (1024, 5)), types: ((tf.int64,
tf.float64), tf.float64)>
targets example
[[323]
 [323]
 [323]
 [323]
 [332]
 [332]
 [332]
 [332]
 [135]
 [135]]
context example (positive context in index 0)
[[ 61. 204. 186. 250. 429.]
 [  5. 417. 171. 235. 175.]
 [  1. 268. 112. 189. 125.]
 [320. 479. 148.  95. 164.]
 [165.  99. 160.   2. 395.]
 [  1. 342. 337. 335. 413.]
 [ 68. 284. 462. 216. 367.]
 [319.  65. 265.  38. 411.]
 [  3. 283. 433.  72. 350.]
 [270. 473. 391. 107. 130.]]
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.]]
```

```
2021-10-30 02:50:46.733504: 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
2021-10-30 02:50:46.841230: 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
2021-10-30 02:50:46.842134: 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
2021-10-30 02:50:46.845823: 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 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-10-30 02:50:46.846123: 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
2021-10-30 02:50:46.846896: 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
2021-10-30 02:50:46.847664: 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
2021-10-30 02:50:48.873251: 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
2021-10-30 02:50:48.874089: 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
2021-10-30 02:50:48.874900: 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
2021-10-30 02:50:48.876510: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 10819 MB memory:  -> device:
0, name: Tesla K80, pci bus id: 0000:00:04.0, compute capability: 3.7
```

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` - 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):
    super(Word2Vec, self).__init__()
    self.target_embedding = None
    self.context_embedding = None

    # YOUR CODE HERE #
    self.target_embedding = tf.keras.layers.Embedding(vocab_size,␣
 ↪embedding_dim, input_length=1)
    self.context_embedding = tf.keras.layers.Embedding(vocab_size,␣
 ↪embedding_dim, input_length=num_ns+1)
    # 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)
    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 >0.9.

[16]:
```
# YOUR CODE HERE #
epochs = 100
embedding_dim = 32
num_ns = 4
model_word2vec = Word2Vec(vocab_size, embedding_dim, num_ns)
model_word2vec.compile(optimizer='adam',
                 loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
                 metrics=['accuracy'])
model_word2vec.fit(dataset, epochs=epochs)
# END CODE #
```

```
Epoch 1/100
```

```
2021-10-30 02:50:49.678312: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)

251/251 [==============================] - 4s 9ms/step - loss: 1.2520 -
accuracy: 0.6738
Epoch 2/100
251/251 [==============================] - 2s 9ms/step - loss: 0.7752 -
accuracy: 0.7206
Epoch 3/100
251/251 [==============================] - 2s 9ms/step - loss: 0.7070 -
accuracy: 0.7412
Epoch 4/100
251/251 [==============================] - 2s 9ms/step - loss: 0.6467 -
accuracy: 0.7657
Epoch 5/100
251/251 [==============================] - 2s 9ms/step - loss: 0.5912 -
accuracy: 0.7882
Epoch 6/100
251/251 [==============================] - 2s 9ms/step - loss: 0.5462 -
accuracy: 0.8055
Epoch 7/100
251/251 [==============================] - 3s 10ms/step - loss: 0.5118 -
accuracy: 0.8174
Epoch 8/100
251/251 [==============================] - 2s 10ms/step - loss: 0.4854 -
accuracy: 0.8266
Epoch 9/100
251/251 [==============================] - 2s 9ms/step - loss: 0.4652 -
accuracy: 0.8342
Epoch 10/100
251/251 [==============================] - 2s 9ms/step - loss: 0.4488 -
accuracy: 0.8399
Epoch 11/100
251/251 [==============================] - 2s 9ms/step - loss: 0.4356 -
accuracy: 0.8445
Epoch 12/100
251/251 [==============================] - 2s 9ms/step - loss: 0.4244 -
accuracy: 0.8481
Epoch 13/100
251/251 [==============================] - 2s 9ms/step - loss: 0.4150 -
accuracy: 0.8513
Epoch 14/100
251/251 [==============================] - 2s 9ms/step - loss: 0.4069 -
accuracy: 0.8542
Epoch 15/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3996 -
accuracy: 0.8569
```

```
Epoch 16/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3934 -
accuracy: 0.8591
Epoch 17/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3877 -
accuracy: 0.8609
Epoch 18/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3826 -
accuracy: 0.8629
Epoch 19/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3779 -
accuracy: 0.8646
Epoch 20/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3738 -
accuracy: 0.8663
Epoch 21/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3699 -
accuracy: 0.8676
Epoch 22/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3665 -
accuracy: 0.8688
Epoch 23/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3633 -
accuracy: 0.8700
Epoch 24/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3601 -
accuracy: 0.8711
Epoch 25/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3574 -
accuracy: 0.8723
Epoch 26/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3549 -
accuracy: 0.8732
Epoch 27/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3523 -
accuracy: 0.8742
Epoch 28/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3500 -
accuracy: 0.8749
Epoch 29/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3477 -
accuracy: 0.8759
Epoch 30/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3458 -
accuracy: 0.8764
Epoch 31/100
251/251 [==============================] - 3s 10ms/step - loss: 0.3438 -
accuracy: 0.8771
```

```
Epoch 32/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3421 -
accuracy: 0.8776
Epoch 33/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3402 -
accuracy: 0.8784
Epoch 34/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3387 -
accuracy: 0.8788
Epoch 35/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3371 -
accuracy: 0.8792
Epoch 36/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3356 -
accuracy: 0.8797
Epoch 37/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3342 -
accuracy: 0.8803
Epoch 38/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3329 -
accuracy: 0.8806
Epoch 39/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3316 -
accuracy: 0.8810
Epoch 40/100
251/251 [==============================] - 2s 10ms/step - loss: 0.3304 -
accuracy: 0.8815
Epoch 41/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3291 -
accuracy: 0.8818
Epoch 42/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3280 -
accuracy: 0.8824
Epoch 43/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3270 -
accuracy: 0.8826
Epoch 44/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3259 -
accuracy: 0.8830
Epoch 45/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3250 -
accuracy: 0.8833
Epoch 46/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3240 -
accuracy: 0.8837
Epoch 47/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3232 -
accuracy: 0.8839
```

```
Epoch 48/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3224 -
accuracy: 0.8841
Epoch 49/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3215 -
accuracy: 0.8845
Epoch 50/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3208 -
accuracy: 0.8848
Epoch 51/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3200 -
accuracy: 0.8849
Epoch 52/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3191 -
accuracy: 0.8852
Epoch 53/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3185 -
accuracy: 0.8855
Epoch 54/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3178 -
accuracy: 0.8856
Epoch 55/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3171 -
accuracy: 0.8859
Epoch 56/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3165 -
accuracy: 0.8861
Epoch 57/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3159 -
accuracy: 0.8863
Epoch 58/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3152 -
accuracy: 0.8865
Epoch 59/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3146 -
accuracy: 0.8867
Epoch 60/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3139 -
accuracy: 0.8870
Epoch 61/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3134 -
accuracy: 0.8872
Epoch 62/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3128 -
accuracy: 0.8874
Epoch 63/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3124 -
accuracy: 0.8876
```

```
Epoch 64/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3119 -
accuracy: 0.8877
Epoch 65/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3115 -
accuracy: 0.8878
Epoch 66/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3109 -
accuracy: 0.8880
Epoch 67/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3105 -
accuracy: 0.8882
Epoch 68/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3100 -
accuracy: 0.8883
Epoch 69/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3097 -
accuracy: 0.8884
Epoch 70/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3091 -
accuracy: 0.8886
Epoch 71/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3088 -
accuracy: 0.8888
Epoch 72/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3083 -
accuracy: 0.8887
Epoch 73/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3080 -
accuracy: 0.8890
Epoch 74/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3076 -
accuracy: 0.8891
Epoch 75/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3072 -
accuracy: 0.8892
Epoch 76/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3067 -
accuracy: 0.8894
Epoch 77/100
251/251 [==============================] - 2s 10ms/step - loss: 0.3065 -
accuracy: 0.8894
Epoch 78/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3062 -
accuracy: 0.8895
Epoch 79/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3058 -
accuracy: 0.8897
```

```
Epoch 80/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3055 -
accuracy: 0.8897
Epoch 81/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3052 -
accuracy: 0.8899
Epoch 82/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3048 -
accuracy: 0.8902
Epoch 83/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3044 -
accuracy: 0.8902
Epoch 84/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3041 -
accuracy: 0.8904
Epoch 85/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3040 -
accuracy: 0.8905
Epoch 86/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3036 -
accuracy: 0.8906
Epoch 87/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3033 -
accuracy: 0.8907
Epoch 88/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3030 -
accuracy: 0.8909
Epoch 89/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3027 -
accuracy: 0.8909
Epoch 90/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3023 -
accuracy: 0.8910
Epoch 91/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3022 -
accuracy: 0.8910
Epoch 92/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3019 -
accuracy: 0.8911
Epoch 93/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3016 -
accuracy: 0.8912
Epoch 94/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3014 -
accuracy: 0.8913
Epoch 95/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3011 -
accuracy: 0.8915
```

```
Epoch 96/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3009 -
accuracy: 0.8915
Epoch 97/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3006 -
accuracy: 0.8917
Epoch 98/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3005 -
accuracy: 0.8917
Epoch 99/100
251/251 [==============================] - 2s 9ms/step - loss: 0.3002 -
accuracy: 0.8919
Epoch 100/100
251/251 [==============================] - 2s 9ms/step - loss: 0.2999 -
accuracy: 0.8920
```

```
[16]: <keras.callbacks.History at 0x7f6606b014d0>
```

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

Written answer: I would expect an accuracy of 20% because there is a 1/5 chance that you randomly guess the positive label correctly

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]: 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:
          embedding (np.array[float,float]): with shape (vocab_size, embedding_dim)
          """
          embeddings = None
```

```
    # YOUR CODE HERE #
    embeddings = np.asarray(model_word2vec.target_embedding.get_weights())
    embeddings = np.squeeze(embeddings, axis=0)
    # END CODE #
    return embeddings

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,501)
    if idx>=VOCAB_SIZE:
        return 'ERROR: NOT IN VOCAB'
    nearest = np.argsort(dists[idx])[::-1]
    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 = ['bleeding','pain','and','off','breath','air',␣
↪'rhythm','started', 'medicine', 'infection']
# END CODE #

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

```
TARGET: bleeding
NEAREST: bleed, infection, gi, been, any, significant, report, fevers, some,
prior, signs, not, while, symptoms,


TARGET: pain
NEAREST: fevers, fever, nausea, breath, vomiting, cough, abdominal, shortness,
prn, constipation, chest, back, or, lower,


TARGET: and
NEAREST: noted, tube, her, however, continued, pleural, so, chest, <unk>, was,
upper, fluid, low, which,


TARGET: off
NEAREST: then, discontinued, so, coumadin, which, morning, every, hours, this,
placement, days, placed, q6h, patient,


TARGET: breath
NEAREST: pain, shortness, chest, some, without, nausea, fever, upper, lungs,
cough, any, signs, bleeding, due,


TARGET: air
NEAREST: wall, sounds, micu, oxygen, ra, room, ct, fluid, size, 90, rhythm, in,
100, the,


TARGET: rhythm
NEAREST: sinus, oxygen, sounds, rate, alert, ekg, size, with, improved,
decreased, echo, in, good, regular,


TARGET: started
NEAREST: when, discontinued, given, initially, placed, recommended, received,
treated, continued, intubated, held, vancomycin, so, followed,
TARGET: medicine
NEAREST: service, sex, m, therapy, inr, after, pt, f, general, need, surgery,
hematocrit, needed, facility,


TARGET: infection
NEAREST: bleeding, all, care, been, acute, time, urine, disease, bleed, level,
symptoms, no, distress, antibiotics,
```

1.2.1 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]:
```python
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,

Written answer: 1. We can use the last 512 word embeddings from each note as its representation, as this might be where the clinician includes the most relevant information of the user's condition and has the highest information value in the note. 2. We can sample the 512 most frequent words as that is most likely the most important words and has highest information value in the note. If some words occur many times in comparison to others, it might be a good indicator that those words are particularly informative and important to the patient's health.

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. - 1 LSTM layer that returns a single vector (instead of a sequence of vectors). - 1 dropout layer. - 1 dense layer whose output is a prediction.

```
[23]: num_timesteps, num_features = X_train.shape[-2:]
num_lstm_units=32
# YOUR CODE HERE #
model_lstm = tf.keras.Sequential(
        layers=[
            tf.keras.layers.Masking(mask_value=0.,
                                    input_shape=(num_timesteps, num_features)),
            tf.keras.layers.LSTM(num_lstm_units),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.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.

Written answer: The LSTM layer returns only the last element in this problem, because it is a "many-to-one" problem. We are interested in predicting a single vector rather than a sequence of vectors as we want to predict which is the positive label. As a result, in contrast to the previous question, we only return the last element.

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.

```
[24]: # YOUR CODE HERE #
model_lstm.compile(loss=tf.keras.losses.BinaryCrossentropy(), optimizer=tf.
 ↪keras.optimizers.Adam(learning_rate=0.0001), metrics=['accuracy'])
epochs = 15
hist = model_lstm.fit(x=X_train, y=y_train, epochs=epochs,␣
 ↪validation_data=(X_val, y_val))
# END CODE #
```

```
Epoch 1/15
18/18 [==============================] - 6s 156ms/step - loss: 0.7174 -
accuracy: 0.5053 - val_loss: 0.7143 - val_accuracy: 0.4684
Epoch 2/15
18/18 [==============================] - 2s 89ms/step - loss: 0.7126 - accuracy:
0.5053 - val_loss: 0.7065 - val_accuracy: 0.4789
Epoch 3/15
18/18 [==============================] - 2s 87ms/step - loss: 0.7149 - accuracy:
0.4544 - val_loss: 0.7032 - val_accuracy: 0.4737
Epoch 4/15
18/18 [==============================] - 2s 87ms/step - loss: 0.7037 - accuracy:
0.5193 - val_loss: 0.7005 - val_accuracy: 0.4947
Epoch 5/15
18/18 [==============================] - 2s 87ms/step - loss: 0.7082 - accuracy:
0.4825 - val_loss: 0.6983 - val_accuracy: 0.5053
Epoch 6/15
18/18 [==============================] - 2s 88ms/step - loss: 0.6913 - accuracy:
0.5404 - val_loss: 0.6972 - val_accuracy: 0.5053
Epoch 7/15
18/18 [==============================] - 2s 87ms/step - loss: 0.7112 - accuracy:
0.5070 - val_loss: 0.6969 - val_accuracy: 0.5105
Epoch 8/15
18/18 [==============================] - 2s 87ms/step - loss: 0.7018 - accuracy:
0.5228 - val_loss: 0.6962 - val_accuracy: 0.5000
Epoch 9/15
18/18 [==============================] - 2s 88ms/step - loss: 0.7006 - accuracy:
0.5158 - val_loss: 0.6959 - val_accuracy: 0.5263
Epoch 10/15
18/18 [==============================] - 2s 88ms/step - loss: 0.6969 - accuracy:
0.5281 - val_loss: 0.6953 - val_accuracy: 0.5211
Epoch 11/15
18/18 [==============================] - 2s 87ms/step - loss: 0.7024 - accuracy:
0.5000 - val_loss: 0.6954 - val_accuracy: 0.5263
Epoch 12/15
18/18 [==============================] - 2s 88ms/step - loss: 0.7055 - accuracy:
```

```
0.4930 - val_loss: 0.6953 - val_accuracy: 0.5263
Epoch 13/15
18/18 [==============================] - 2s 87ms/step - loss: 0.7098 - accuracy:
0.4789 - val_loss: 0.6951 - val_accuracy: 0.5263
Epoch 14/15
18/18 [==============================] - 2s 88ms/step - loss: 0.6899 - accuracy:
0.5368 - val_loss: 0.6951 - val_accuracy: 0.5211
Epoch 15/15
18/18 [==============================] - 2s 88ms/step - loss: 0.6946 - accuracy:
0.5158 - val_loss: 0.6948 - val_accuracy: 0.5263
```

Q2.13

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

Written answer: 1. The validation set may have words we have not yet learned embeddings for. As a result, the model cannot accurately predict on this new dataset, since we don't have the learned values to predict on - this may result in poor predictive efficiency on the validation test set. 2. Inter-clinician variability may also be cause for the poor validation result. As we know, doctors differ greatly in how they write and append their patient notes, which may have accounted for the poor predictive quality on the validation test set.

1.3

A2_part3_clinical_BERT_embeddings_and_readmission_prediction

October 30, 2021

1 Assignment 2 - part 3 - BERT Embeddings For Prediction

In this part of the assignment, we will attempt the same prediction task as part 2, but with two differences.

Different subsequencing strategy Models for sequence data need fixed sequence lengths. In part 2 we just used just the first ~500 words of each note. In part 3 we will break each note into 500-word chunks and train the model to classify each chunk separately. Then we will combine the chunked predictions into one prediction for the whole note. This is sometimes referred to as a 'sliding window' or 'binning'. (Here is a discussion of strategies for long-text modeling with BERT.)

Different embedding strategy We need to convert sequences of word tokens to a vector representation that we can then use in a prediction model. In part 2 we converted each of the first 500 words into 500 Word2Vec embedding vectors, and then passed that sequence of 500 vectors to an LSTM prediction model. In part 3 we will instead convert each note sequence to a single vector. This vector is something we can get from BERT, a popular transformer model. Specifically we will be using a BERT model trained on biomedical and clinical data, similar to the ClinicalBert paper.

In the next cell replace `ROOT` with your path.

[1]:
```
import readmission_utils
import tensorflow as tf
import pandas as pd
import random
import pickle
import numpy as np
import matplotlib.pyplot as plt
import bert_utils

ROOT = "/home/marchuo/assign2/"    # Put your root path here"
tf.keras.backend.set_floatx('float32')
```

1.1 Preprocessing text data and visualization

Execute the code in the next cell, which will take about 60mins the first time you run it. It will save its results to a file in `ROOT/saved_data/texts_to_labels_5000.pkl`. This is the same code as part 2, except now we have 5000 notes instead of 1000.

If the file already exists then calling the function will just load the results. We also break the notes and labels into train/val/test sets.

```
[2]: notes, labels = readmission_utils.get_notes_and_labels(ROOT, 5000)
```

```
Found file /home/marchuo/assign2/saved_data/texts_to_labels_5000.pkl, loading
```

Run the following code which loads a a pretrained Bert model from the HuggingFace transformers library. This library provides a standard interface for tokenizers and transformers in Tensorflow and PyTorch. HuggingFace also provides a platform for reserachers to share pretrained models. For example we are using this BERT model + tokenizer that has been trained on a dataset of biomedical texts.

This code should take less than 1 minute to run.

```
[3]: from transformers import AutoTokenizer, TFAutoModel
     import readmission_utils
     hf_model = "cambridgeltl/SapBERT-from-PubMedBERT-fulltext"
     if 'tokenizer' not in locals().keys(): tokenizer = AutoTokenizer.
      ↪from_pretrained(hf_model)
     if 'bert_auto_model' not in locals().keys(): bert_model = TFAutoModel.
      ↪from_pretrained(hf_model)
     tokenizer = AutoTokenizer.from_pretrained(hf_model)
     bert_model = TFAutoModel.from_pretrained(hf_model)
```

```
2021-10-30 03:36:39.101694: 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
2021-10-30 03:36:39.110661: 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
2021-10-30 03:36:39.111448: 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
2021-10-30 03:36:39.113495: 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 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-10-30 03:36:39.113818: 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
2021-10-30 03:36:39.114531: 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
2021-10-30 03:36:39.115303: 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
2021-10-30 03:36:39.508666: 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
2021-10-30 03:36:39.509580: 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
2021-10-30 03:36:39.510356: 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
2021-10-30 03:36:39.511049: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 10819 MB memory:  -> device:
0, name: Tesla K80, pci bus id: 0000:00:04.0, compute capability: 3.7
All model checkpoint layers were used when initializing TFBertModel.

All the layers of TFBertModel were initialized from the model checkpoint at
cambridgeltl/SapBERT-from-PubMedBERT-fulltext.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use TFBertModel for predictions without further training.
All model checkpoint layers were used when initializing TFBertModel.

All the layers of TFBertModel were initialized from the model checkpoint at
cambridgeltl/SapBERT-from-PubMedBERT-fulltext.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use TFBertModel for predictions without further training.
```

Now run the following data preparation code. We'll explain what it does later. It will take about ~20mins the first time it's run and it will save data to `{ROOT}/saved_data/bert_datasets.pkl`. For later runs, it will just load this file.

[4]:
```
data = bert_utils.prepare_bert_datasets(ROOT, notes, labels, bert_model,␣
 ↪tokenizer)
```

```
File /home/marchuo/assign2//saved_data/bert_datasets.pkl exists.
Loading it
```

Q3.1 BERT architecture

LSTMs model sequence dependencies using recurrence; they are recurrent neural networks or RNNs. In RNNs we pass elements of a sequence through the model one a time (sequentially). Each pass through the RNN updates an internal state vector. Future passes through the RNN are a function of the state vector. This is how RNNs can model dependencies between elements in a sequence.

On the other hand, Bert has a transformer architecture. Transformers are state-of-the-art in most standard tasks in language modelling. Instead of processing sequence data one-at-a-time, transformers process entire sequences at once. But they still model dependencies between sequence elements. Briefly describe the mechanism that transformers use to model sequence dependencies. (You can refer to the lecture slides, or the major transformers paper, Attention is all you need).

Written Answer: Transformers follow the overall architecture of neural sequence transduction models, where they have an encoder-decoder structure. The encoder maps an input sequence of symbol representations to a sequence of continuous representations. Then, the decoder takes this sequence of continuous representations and generates an output sequence of symbols one at a time. The encoder-decoder structure is auto-regressive, consuming previously generated symbols. The transformer uses fully stacked self-attention and point wise layers.

The encoder has 6 layers with 2 sublayers between each - one is a multi-head self-attention mechanism and the other is a fully connected position-wise feed-forward network. The decoder has 6 layers with 1 sublayer in addition to the two aforementioned sublayers (total 3), which performs multi-head attention. The self-attention layer is masked by offsetting by 1 position and preventing positions from attending to subsequent positions, so that "the predictions for position i can depend only on the known outputs at positions less than i".

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. Thus, every position in the decoder is able to attend over all positions in the input sequence. Similarly, each position in the encoder can attend to all positions in the previous layer of the encoder. Finally, every position in the decoder is able to attend over all positions in the decoder sequences up until that point.

Q3.2 BERT pretraining

Briefly describe the 2 pretraining tasks discussed in the introduction to the BERT paper.

Written Answer: 1) Masked LM. The researchers masked some percentage of the input tokens and predicted those masked tokens. The "final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary". In this particular paper, they masked 15% of the WordPiece tokens in each sequence at random, and they predicted the missing words. This allows them to receive a bidirectional pre-trained model. They also account for mismatch between pre-training and finetuning by varying the replacements of the tokens.

2) Next Sentence Prediction (NSP). The researchers pre-trained for a binarized next sentence prediction task. When choosing sentences A and B for each pre-training example, 50% of the time B is the sentence that follows A or 50% of the time a random sentence is the sentence that follows A. This is especially beneficial for Question Answering and Natural Language Inference.

Q3.3 datasets for BERT pretraining

What is the benefit of using a BERT model that has been pretrained on biomedical text compared with, for example, a BERT model trained on Wikipedia?

Written Answer: By using a BERT model that has been pretrained on biomedical text, the model has been trained on word embeddings that are specifically referenced on clinician notes and is incredibly relevant to our prediction task, which is biomedical in nature. On the other hand, if we were to use a BERT model train on Wikipedia, there would be a myriad of words that are never referenced and seen in the notes corpus, and the model would also be much less likely to have biomedical terms referenced in our clinician notes. There would be an abundance of unused words if trained on Wikipedia

Q3.4 data chunking strategy

Let's look at some of the data we created earlier when we ran `bert_utils.prepare_bert_datasets`. First we did a train/val/test split for the notes and labels. All these variables have the suffix, `_FULL`, indicating that this is the full note, before chunking.

[5]:
```
[
    train_notes_FULL, train_labels_FULL,
    val_notes_FULL, val_labels_FULL,
    test_notes_FULL, test_labels_FULL
] = data['FULL']

all_note_lengths = [len(train_notes_FULL), len(test_notes_FULL),␣
 ↪len(test_labels_FULL)]
print(f"train/val/test lengths {all_note_lengths} \n")
print(f"Which sum to {sum(all_note_lengths)}")
print(f"Original notes len {len(notes)}")
```

```
train/val/test lengths [2853, 951, 951]

Which sum to 4755
Original notes len 4755
```

Now we do chunking for the train, val and test sets separately (this is what we described in the "Different subsequencing strategy" section at the start of the assignment).

Take the train set for example. We break the notes into ~500 word chunks, `train_notes_CHUNKS`. We copy the labels into `train_labels_CHUNKS`. Finally, `train_idxs_CHUNKS` tells you which FULL note this CHUNK is from. Suppose `train_idxs_CHUNKS[30]=6`; this means `train_notes_CHUNKS[30]` is a subsequence of the note `train_notes_FULL[6]`.

Here is an example: - If `train_notes_FULL[0]` is about 1200 words long, then we create 3 note-chunks that will be in `train_notes_CHUNKS[0:3]` - If the label is `train_notes_FULL[0]=1` then we copy that label for each note-chunk, so `train_labels_CHUNKS[0:3]=1`. - Since these chunks are all subsequences of `train_notes_FULL[0]`, we set `train_idxs_CHUNKS[0:3]=0`.

We print the labels and indxs for the first 25 entries. You should verify that the results match your understanding of this dataset.

[6]:
```
[
    train_notes_CHUNKS, train_labels_CHUNKS, train_idxs_CHUNKS,
    val_notes_CHUNKS, val_labels_CHUNKS, val_idxs_CHUNKS,
    test_notes_CHUNKS, test_labels_CHUNKS, test_idxs_CHUNKS,
] = data['CHUNKS_DATA']
```

```
print("First 20 chunks:")
print(f"Labels      : {train_labels_CHUNKS[:25]}")
print(f"Indexes.    : {train_idxs_CHUNKS[:25]}")
```

```
First 20 chunks:
Labels      : [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
0, 0, 0]
Indexes.    : [0 0 0 0 0 0 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 4 4 4 4]
```

Briefly discuss the pros and cons of this data chunking strategy compared to the truncation strategy used in part 2.

Written answer: One of the strengths of this data chunking strategy rather than truncation is that we have more context for the note because you get bert embeddings for the whole note rather than just target words as compared to part 2. As a result, we can get a more holistic overview of the note rather than just the truncated words. Some of the cons could be that this process is more computationally expensive and time-consuming to accomplish in comparison to the truncated strategy. In addition, we'll have more imbalance between very frequent and common words in comparison to rare words, which may affect model accuracy.

Q3.5 BERT embeddings

Finally we took these chunked notes and put them into BERT pooled embeddings.

[7]:
```
[
    train_bert_pool_embeddings_CHUNKS,
    val_bert_pool_embeddings_CHUNKS,
    test_bert_pool_embeddings_CHUNKS,
] = data['CHUNKS_EMEDDINGS']
print(f"Length of train_notes_CHUNKS                {len(train_notes_CHUNKS)}")
print(f"Shape of train_bert_pool_embeddings_CHUNKS␣
 ↪{train_bert_pool_embeddings_CHUNKS.shape}")
```

```
Length of train_notes_CHUNKS                12253
Shape of train_bert_pool_embeddings_CHUNKS (12253, 768)
```

2 For our prediction task:

- The x-data is `train_bert_pool_embeddings_CHUNKS`.
- They y-labels are `train_labels_CHUNKS`.

Look at the shape of `train_bert_pool_embeddings_CHUNKS` printed in the above cell. There is one single BERT embedding vector for each note chunk. This is called the "pooled BERT embedding", and is also the h_{CLS} token output discussed in lecture. s How is this embedding different to the embeddings used in part 2? Specifically talk about the shape of the data that we will pass into a prediction model.

Written answer: In part 2, we had a 32 dimension embedding, while we now have embeddings of size 768. There is one single BERT embedding vector for each note chunk, while the previous part embeds each word into a vector representation of size 32.

2.0.1 Prediction model

Q3.6 build and run prediction model The inputs to our model are single-vector BERT embeddings. These embeddings should do a very good job of summarising the text, such that our prediction model can be extremely simple: - The input is the BERT embedding vector. - We have one Dense layer with 1 node output and sigmoid activation (no hidden layers).

Compile this model with. - Adam. - Binary cross entropy loss. - Metrics for accuracy and AUC.

Train it for 100 epoch with batch size 128, and pass in the validation dataset.

(Optional: you can experiment with adding extra dense layers and dropout. See if you can avoid overfitting.)

[19]:
```
train_x = train_bert_pool_embeddings_CHUNKS
train_y = np.array(train_labels_CHUNKS)
val_x = val_bert_pool_embeddings_CHUNKS
val_y = np.array(val_labels_CHUNKS)
test_x = test_bert_pool_embeddings_CHUNKS
test_y = np.array(test_labels_CHUNKS)

# YOUR CODE HERE #
model = tf.keras.Sequential(
        layers=[
            tf.keras.layers.Dense(1, activation='sigmoid')
        ]
    )
model.compile(loss=tf.keras.losses.BinaryCrossentropy(), optimizer=tf.keras.
 ↪optimizers.Adam(learning_rate=0.0001), metrics=['accuracy', tf.keras.metrics.
 ↪AUC()])
epochs = 100
batch_size = 128
hist = model.fit(x=train_x, y=train_y, epochs=epochs, validation_data=(val_x,␣
 ↪val_y), batch_size=batch_size)
# END CODE #
```

```
Epoch 1/100
96/96 [==============================] - 1s 7ms/step - loss: 0.7046 - accuracy:
0.5085 - auc_2: 0.5062 - val_loss: 0.7047 - val_accuracy: 0.5038 - val_auc_2:
0.5024
Epoch 2/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6989 - accuracy:
0.5139 - auc_2: 0.5154 - val_loss: 0.6983 - val_accuracy: 0.5128 - val_auc_2:
0.5161
Epoch 3/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6949 - accuracy:
0.5247 - auc_2: 0.5267 - val_loss: 0.6947 - val_accuracy: 0.5227 - val_auc_2:
0.5311
```

```
Epoch 4/100
96/96 [==============================] - 1s 5ms/step - loss: 0.6918 - accuracy:
0.5360 - auc_2: 0.5400 - val_loss: 0.6914 - val_accuracy: 0.5315 - val_auc_2:
0.5444
Epoch 5/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6893 - accuracy:
0.5399 - auc_2: 0.5498 - val_loss: 0.6890 - val_accuracy: 0.5405 - val_auc_2:
0.5578
Epoch 6/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6873 - accuracy:
0.5417 - auc_2: 0.5594 - val_loss: 0.6864 - val_accuracy: 0.5521 - val_auc_2:
0.5690
Epoch 7/100
96/96 [==============================] - 0s 4ms/step - loss: 0.6855 - accuracy:
0.5470 - auc_2: 0.5675 - val_loss: 0.6846 - val_accuracy: 0.5582 - val_auc_2:
0.5779
Epoch 8/100
96/96 [==============================] - 1s 6ms/step - loss: 0.6836 - accuracy:
0.5487 - auc_2: 0.5745 - val_loss: 0.6826 - val_accuracy: 0.5631 - val_auc_2:
0.5853
Epoch 9/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6823 - accuracy:
0.5550 - auc_2: 0.5800 - val_loss: 0.6818 - val_accuracy: 0.5638 - val_auc_2:
0.5917
Epoch 10/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6810 - accuracy:
0.5566 - auc_2: 0.5850 - val_loss: 0.6805 - val_accuracy: 0.5674 - val_auc_2:
0.5975
Epoch 11/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6798 - accuracy:
0.5606 - auc_2: 0.5897 - val_loss: 0.6800 - val_accuracy: 0.5648 - val_auc_2:
0.6019
Epoch 12/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6792 - accuracy:
0.5634 - auc_2: 0.5916 - val_loss: 0.6781 - val_accuracy: 0.5721 - val_auc_2:
0.6054
Epoch 13/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6781 - accuracy:
0.5652 - auc_2: 0.5952 - val_loss: 0.6764 - val_accuracy: 0.5806 - val_auc_2:
0.6090
Epoch 14/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6777 - accuracy:
0.5648 - auc_2: 0.5955 - val_loss: 0.6767 - val_accuracy: 0.5762 - val_auc_2:
0.6115
Epoch 15/100
96/96 [==============================] - 1s 5ms/step - loss: 0.6767 - accuracy:
0.5683 - auc_2: 0.5998 - val_loss: 0.6747 - val_accuracy: 0.5818 - val_auc_2:
0.6141
```

```
Epoch 16/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6764 - accuracy:
0.5715 - auc_2: 0.6004 - val_loss: 0.6752 - val_accuracy: 0.5823 - val_auc_2:
0.6160
Epoch 17/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6756 - accuracy:
0.5710 - auc_2: 0.6021 - val_loss: 0.6744 - val_accuracy: 0.5830 - val_auc_2:
0.6177
Epoch 18/100
96/96 [==============================] - 1s 5ms/step - loss: 0.6746 - accuracy:
0.5737 - auc_2: 0.6062 - val_loss: 0.6729 - val_accuracy: 0.5917 - val_auc_2:
0.6196
Epoch 19/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6745 - accuracy:
0.5745 - auc_2: 0.6056 - val_loss: 0.6733 - val_accuracy: 0.5852 - val_auc_2:
0.6207
Epoch 20/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6740 - accuracy:
0.5773 - auc_2: 0.6076 - val_loss: 0.6725 - val_accuracy: 0.5876 - val_auc_2:
0.6220
Epoch 21/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6736 - accuracy:
0.5755 - auc_2: 0.6086 - val_loss: 0.6717 - val_accuracy: 0.5920 - val_auc_2:
0.6236
Epoch 22/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6731 - accuracy:
0.5765 - auc_2: 0.6100 - val_loss: 0.6721 - val_accuracy: 0.5883 - val_auc_2:
0.6243
Epoch 23/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6732 - accuracy:
0.5766 - auc_2: 0.6092 - val_loss: 0.6708 - val_accuracy: 0.5951 - val_auc_2:
0.6254
Epoch 24/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6724 - accuracy:
0.5772 - auc_2: 0.6114 - val_loss: 0.6714 - val_accuracy: 0.5888 - val_auc_2:
0.6259
Epoch 25/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6720 - accuracy:
0.5783 - auc_2: 0.6127 - val_loss: 0.6719 - val_accuracy: 0.5866 - val_auc_2:
0.6272
Epoch 26/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6718 - accuracy:
0.5817 - auc_2: 0.6134 - val_loss: 0.6711 - val_accuracy: 0.5903 - val_auc_2:
0.6280
Epoch 27/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6714 - accuracy:
0.5794 - auc_2: 0.6144 - val_loss: 0.6713 - val_accuracy: 0.5900 - val_auc_2:
0.6283
```

```
Epoch 28/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6712 - accuracy:
0.5791 - auc_2: 0.6151 - val_loss: 0.6695 - val_accuracy: 0.5966 - val_auc_2:
0.6290
Epoch 29/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6706 - accuracy:
0.5810 - auc_2: 0.6166 - val_loss: 0.6694 - val_accuracy: 0.5949 - val_auc_2:
0.6301
Epoch 30/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6705 - accuracy:
0.5784 - auc_2: 0.6170 - val_loss: 0.6688 - val_accuracy: 0.5988 - val_auc_2:
0.6304
Epoch 31/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6702 - accuracy:
0.5820 - auc_2: 0.6178 - val_loss: 0.6686 - val_accuracy: 0.5966 - val_auc_2:
0.6309
Epoch 32/100
96/96 [==============================] - 1s 5ms/step - loss: 0.6698 - accuracy:
0.5837 - auc_2: 0.6187 - val_loss: 0.6686 - val_accuracy: 0.5990 - val_auc_2:
0.6313
Epoch 33/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6696 - accuracy:
0.5821 - auc_2: 0.6193 - val_loss: 0.6690 - val_accuracy: 0.5932 - val_auc_2:
0.6316
Epoch 34/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6694 - accuracy:
0.5806 - auc_2: 0.6195 - val_loss: 0.6695 - val_accuracy: 0.5988 - val_auc_2:
0.6319
Epoch 35/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6692 - accuracy:
0.5829 - auc_2: 0.6199 - val_loss: 0.6682 - val_accuracy: 0.5966 - val_auc_2:
0.6325
Epoch 36/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6688 - accuracy:
0.5825 - auc_2: 0.6215 - val_loss: 0.6684 - val_accuracy: 0.5983 - val_auc_2:
0.6328
Epoch 37/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6685 - accuracy:
0.5852 - auc_2: 0.6224 - val_loss: 0.6676 - val_accuracy: 0.5985 - val_auc_2:
0.6330
Epoch 38/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6682 - accuracy:
0.5859 - auc_2: 0.6229 - val_loss: 0.6695 - val_accuracy: 0.5939 - val_auc_2:
0.6332
Epoch 39/100
96/96 [==============================] - 1s 5ms/step - loss: 0.6681 - accuracy:
0.5841 - auc_2: 0.6234 - val_loss: 0.6681 - val_accuracy: 0.6015 - val_auc_2:
0.6337
```

```
Epoch 40/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6679 - accuracy:
0.5843 - auc_2: 0.6236 - val_loss: 0.6679 - val_accuracy: 0.5993 - val_auc_2:
0.6342
Epoch 41/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6676 - accuracy:
0.5854 - auc_2: 0.6244 - val_loss: 0.6694 - val_accuracy: 0.5944 - val_auc_2:
0.6341
Epoch 42/100
96/96 [==============================] - 1s 5ms/step - loss: 0.6676 - accuracy:
0.5841 - auc_2: 0.6248 - val_loss: 0.6666 - val_accuracy: 0.5983 - val_auc_2:
0.6349
Epoch 43/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6674 - accuracy:
0.5870 - auc_2: 0.6253 - val_loss: 0.6690 - val_accuracy: 0.5949 - val_auc_2:
0.6348
Epoch 44/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6673 - accuracy:
0.5870 - auc_2: 0.6250 - val_loss: 0.6679 - val_accuracy: 0.5985 - val_auc_2:
0.6349
Epoch 45/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6669 - accuracy:
0.5868 - auc_2: 0.6264 - val_loss: 0.6666 - val_accuracy: 0.5971 - val_auc_2:
0.6354
Epoch 46/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6669 - accuracy:
0.5863 - auc_2: 0.6259 - val_loss: 0.6664 - val_accuracy: 0.5976 - val_auc_2:
0.6358
Epoch 47/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6666 - accuracy:
0.5877 - auc_2: 0.6271 - val_loss: 0.6682 - val_accuracy: 0.5959 - val_auc_2:
0.6353
Epoch 48/100
96/96 [==============================] - 0s 4ms/step - loss: 0.6670 - accuracy:
0.5870 - auc_2: 0.6261 - val_loss: 0.6663 - val_accuracy: 0.5988 - val_auc_2:
0.6359
Epoch 49/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6663 - accuracy:
0.5882 - auc_2: 0.6280 - val_loss: 0.6659 - val_accuracy: 0.6002 - val_auc_2:
0.6364
Epoch 50/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6662 - accuracy:
0.5866 - auc_2: 0.6279 - val_loss: 0.6663 - val_accuracy: 0.5983 - val_auc_2:
0.6364
Epoch 51/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6658 - accuracy:
0.5867 - auc_2: 0.6291 - val_loss: 0.6675 - val_accuracy: 0.5978 - val_auc_2:
0.6363
```

```
Epoch 52/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6656 - accuracy:
0.5906 - auc_2: 0.6294 - val_loss: 0.6665 - val_accuracy: 0.5995 - val_auc_2:
0.6368
Epoch 53/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6656 - accuracy:
0.5898 - auc_2: 0.6296 - val_loss: 0.6669 - val_accuracy: 0.5966 - val_auc_2:
0.6371
Epoch 54/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6654 - accuracy:
0.5893 - auc_2: 0.6300 - val_loss: 0.6676 - val_accuracy: 0.5951 - val_auc_2:
0.6370
Epoch 55/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6652 - accuracy:
0.5916 - auc_2: 0.6311 - val_loss: 0.6663 - val_accuracy: 0.6007 - val_auc_2:
0.6370
Epoch 56/100
96/96 [==============================] - 1s 5ms/step - loss: 0.6650 - accuracy:
0.5906 - auc_2: 0.6312 - val_loss: 0.6655 - val_accuracy: 0.5995 - val_auc_2:
0.6370
Epoch 57/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6647 - accuracy:
0.5920 - auc_2: 0.6318 - val_loss: 0.6660 - val_accuracy: 0.5998 - val_auc_2:
0.6371
Epoch 58/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6646 - accuracy:
0.5950 - auc_2: 0.6319 - val_loss: 0.6658 - val_accuracy: 0.6000 - val_auc_2:
0.6372
Epoch 59/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6646 - accuracy:
0.5903 - auc_2: 0.6323 - val_loss: 0.6663 - val_accuracy: 0.6000 - val_auc_2:
0.6375
Epoch 60/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6644 - accuracy:
0.5953 - auc_2: 0.6326 - val_loss: 0.6655 - val_accuracy: 0.5993 - val_auc_2:
0.6375
Epoch 61/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6642 - accuracy:
0.5915 - auc_2: 0.6329 - val_loss: 0.6678 - val_accuracy: 0.5939 - val_auc_2:
0.6376
Epoch 62/100
96/96 [==============================] - 0s 4ms/step - loss: 0.6639 - accuracy:
0.5928 - auc_2: 0.6339 - val_loss: 0.6647 - val_accuracy: 0.6007 - val_auc_2:
0.6382
Epoch 63/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6641 - accuracy:
0.5922 - auc_2: 0.6335 - val_loss: 0.6647 - val_accuracy: 0.6015 - val_auc_2:
0.6381
```

```
Epoch 64/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6639 - accuracy:
0.5921 - auc_2: 0.6338 - val_loss: 0.6646 - val_accuracy: 0.6015 - val_auc_2:
0.6380
Epoch 65/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6637 - accuracy:
0.5944 - auc_2: 0.6344 - val_loss: 0.6657 - val_accuracy: 0.6002 - val_auc_2:
0.6382
Epoch 66/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6638 - accuracy:
0.5929 - auc_2: 0.6339 - val_loss: 0.6670 - val_accuracy: 0.5959 - val_auc_2:
0.6381
Epoch 67/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6635 - accuracy:
0.5918 - auc_2: 0.6344 - val_loss: 0.6647 - val_accuracy: 0.6024 - val_auc_2:
0.6384
Epoch 68/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6632 - accuracy:
0.5941 - auc_2: 0.6361 - val_loss: 0.6643 - val_accuracy: 0.6012 - val_auc_2:
0.6384
Epoch 69/100
96/96 [==============================] - 0s 4ms/step - loss: 0.6632 - accuracy:
0.5948 - auc_2: 0.6360 - val_loss: 0.6654 - val_accuracy: 0.5998 - val_auc_2:
0.6383
Epoch 70/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6630 - accuracy:
0.5950 - auc_2: 0.6359 - val_loss: 0.6648 - val_accuracy: 0.6002 - val_auc_2:
0.6385
Epoch 71/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6629 - accuracy:
0.5942 - auc_2: 0.6363 - val_loss: 0.6656 - val_accuracy: 0.6012 - val_auc_2:
0.6385
Epoch 72/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6628 - accuracy:
0.5951 - auc_2: 0.6364 - val_loss: 0.6656 - val_accuracy: 0.5998 - val_auc_2:
0.6385
Epoch 73/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6626 - accuracy:
0.5943 - auc_2: 0.6369 - val_loss: 0.6680 - val_accuracy: 0.5932 - val_auc_2:
0.6386
Epoch 74/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6626 - accuracy:
0.5945 - auc_2: 0.6367 - val_loss: 0.6655 - val_accuracy: 0.6002 - val_auc_2:
0.6387
Epoch 75/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6625 - accuracy:
0.5957 - auc_2: 0.6374 - val_loss: 0.6654 - val_accuracy: 0.6015 - val_auc_2:
0.6390
```

```
Epoch 76/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6624 - accuracy:
0.5950 - auc_2: 0.6375 - val_loss: 0.6640 - val_accuracy: 0.6012 - val_auc_2:
0.6393
Epoch 77/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6625 - accuracy:
0.5959 - auc_2: 0.6373 - val_loss: 0.6654 - val_accuracy: 0.6015 - val_auc_2:
0.6390
Epoch 78/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6623 - accuracy:
0.5930 - auc_2: 0.6380 - val_loss: 0.6646 - val_accuracy: 0.6019 - val_auc_2:
0.6396
Epoch 79/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6621 - accuracy:
0.5952 - auc_2: 0.6376 - val_loss: 0.6642 - val_accuracy: 0.6007 - val_auc_2:
0.6394
Epoch 80/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6618 - accuracy:
0.5963 - auc_2: 0.6383 - val_loss: 0.6657 - val_accuracy: 0.5988 - val_auc_2:
0.6388
Epoch 81/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6618 - accuracy:
0.5942 - auc_2: 0.6387 - val_loss: 0.6642 - val_accuracy: 0.6005 - val_auc_2:
0.6391
Epoch 82/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6622 - accuracy:
0.5972 - auc_2: 0.6376 - val_loss: 0.6657 - val_accuracy: 0.5988 - val_auc_2:
0.6390
Epoch 83/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6617 - accuracy:
0.5944 - auc_2: 0.6393 - val_loss: 0.6638 - val_accuracy: 0.6019 - val_auc_2:
0.6397
Epoch 84/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6615 - accuracy:
0.5970 - auc_2: 0.6392 - val_loss: 0.6640 - val_accuracy: 0.5995 - val_auc_2:
0.6397
Epoch 85/100
96/96 [==============================] - 0s 4ms/step - loss: 0.6615 - accuracy:
0.5964 - auc_2: 0.6395 - val_loss: 0.6646 - val_accuracy: 0.6027 - val_auc_2:
0.6394
Epoch 86/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6614 - accuracy:
0.5983 - auc_2: 0.6399 - val_loss: 0.6652 - val_accuracy: 0.5998 - val_auc_2:
0.6393
Epoch 87/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6613 - accuracy:
0.5963 - auc_2: 0.6398 - val_loss: 0.6638 - val_accuracy: 0.6005 - val_auc_2:
0.6397
```

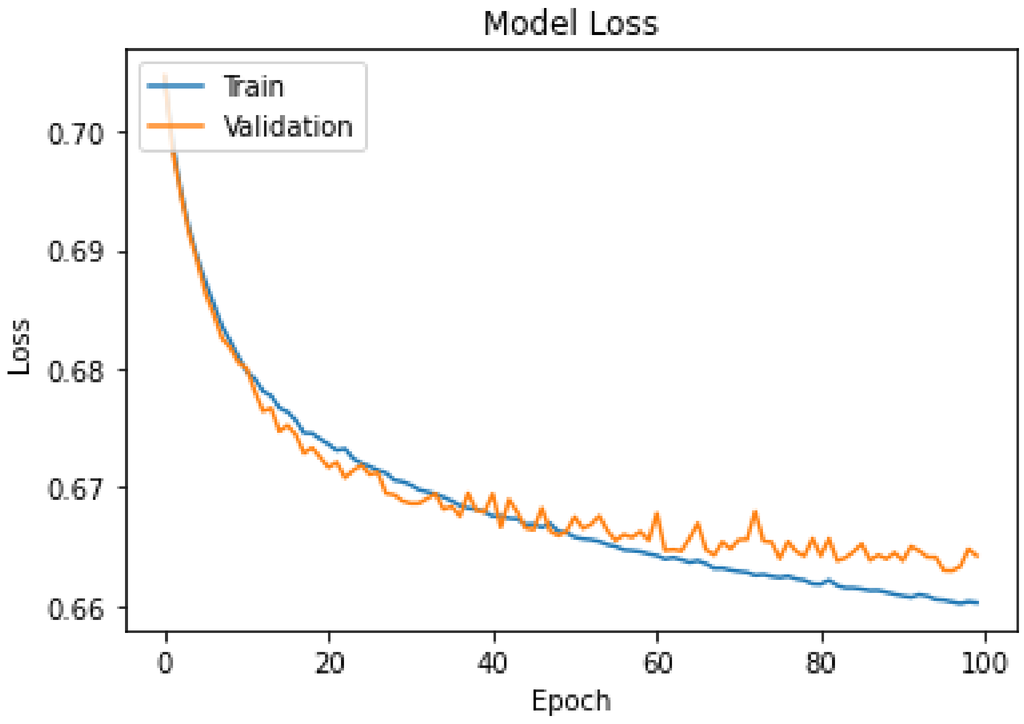
```
Epoch 88/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6613 - accuracy:
0.5950 - auc_2: 0.6401 - val_loss: 0.6643 - val_accuracy: 0.6010 - val_auc_2:
0.6396
Epoch 89/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6611 - accuracy:
0.5975 - auc_2: 0.6403 - val_loss: 0.6639 - val_accuracy: 0.5995 - val_auc_2:
0.6397
Epoch 90/100
96/96 [==============================] - 0s 4ms/step - loss: 0.6610 - accuracy:
0.5981 - auc_2: 0.6409 - val_loss: 0.6645 - val_accuracy: 0.6012 - val_auc_2:
0.6394
Epoch 91/100
96/96 [==============================] - 0s 4ms/step - loss: 0.6608 - accuracy:
0.5963 - auc_2: 0.6410 - val_loss: 0.6638 - val_accuracy: 0.5995 - val_auc_2:
0.6399
Epoch 92/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6607 - accuracy:
0.5981 - auc_2: 0.6415 - val_loss: 0.6650 - val_accuracy: 0.6010 - val_auc_2:
0.6398
Epoch 93/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6609 - accuracy:
0.5983 - auc_2: 0.6405 - val_loss: 0.6646 - val_accuracy: 0.6015 - val_auc_2:
0.6400
Epoch 94/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6608 - accuracy:
0.5979 - auc_2: 0.6411 - val_loss: 0.6641 - val_accuracy: 0.6010 - val_auc_2:
0.6398
Epoch 95/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6605 - accuracy:
0.5976 - auc_2: 0.6417 - val_loss: 0.6641 - val_accuracy: 0.6019 - val_auc_2:
0.6400
Epoch 96/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6605 - accuracy:
0.5976 - auc_2: 0.6416 - val_loss: 0.6630 - val_accuracy: 0.6019 - val_auc_2:
0.6401
Epoch 97/100
96/96 [==============================] - 0s 4ms/step - loss: 0.6603 - accuracy:
0.5989 - auc_2: 0.6420 - val_loss: 0.6630 - val_accuracy: 0.6034 - val_auc_2:
0.6400
Epoch 98/100
96/96 [==============================] - 1s 5ms/step - loss: 0.6602 - accuracy:
0.5972 - auc_2: 0.6425 - val_loss: 0.6633 - val_accuracy: 0.6029 - val_auc_2:
0.6400
Epoch 99/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6603 - accuracy:
0.5972 - auc_2: 0.6421 - val_loss: 0.6648 - val_accuracy: 0.5998 - val_auc_2:
0.6398
```

```
Epoch 100/100
96/96 [==============================] - 0s 5ms/step - loss: 0.6603 - accuracy:
0.5984 - auc_2: 0.6426 - val_loss: 0.6642 - val_accuracy: 0.6022 - val_auc_2:
0.6401
```

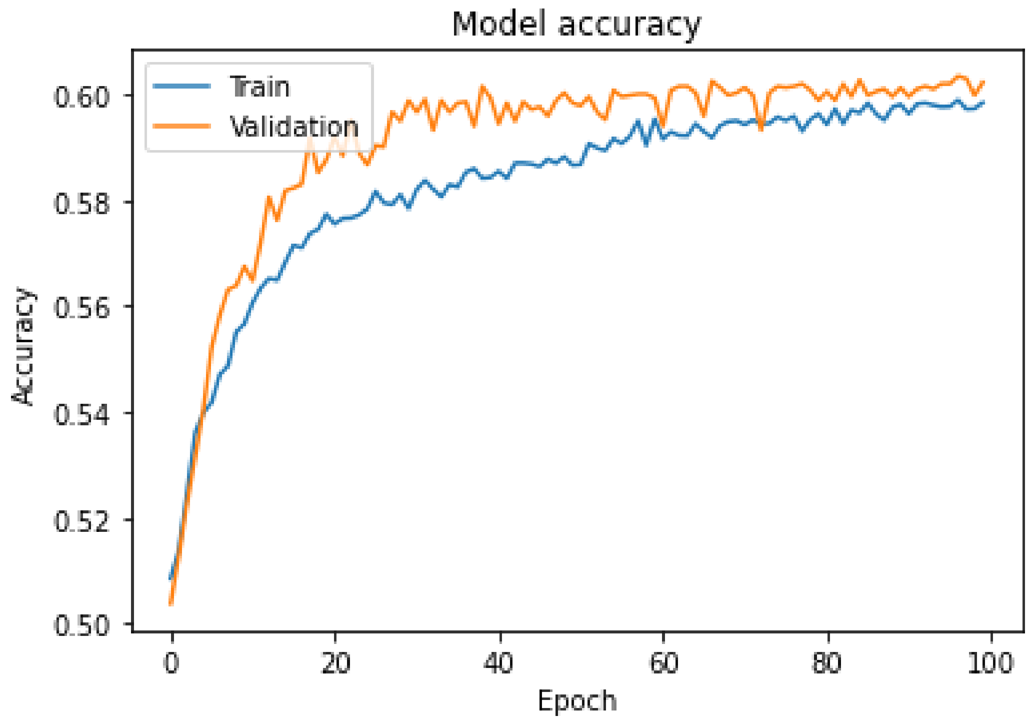
Q3.7 assessing model performance

Make 3 plots: one each for loss, accuracy and AUC. Each plot should have train and validation scores labeled.

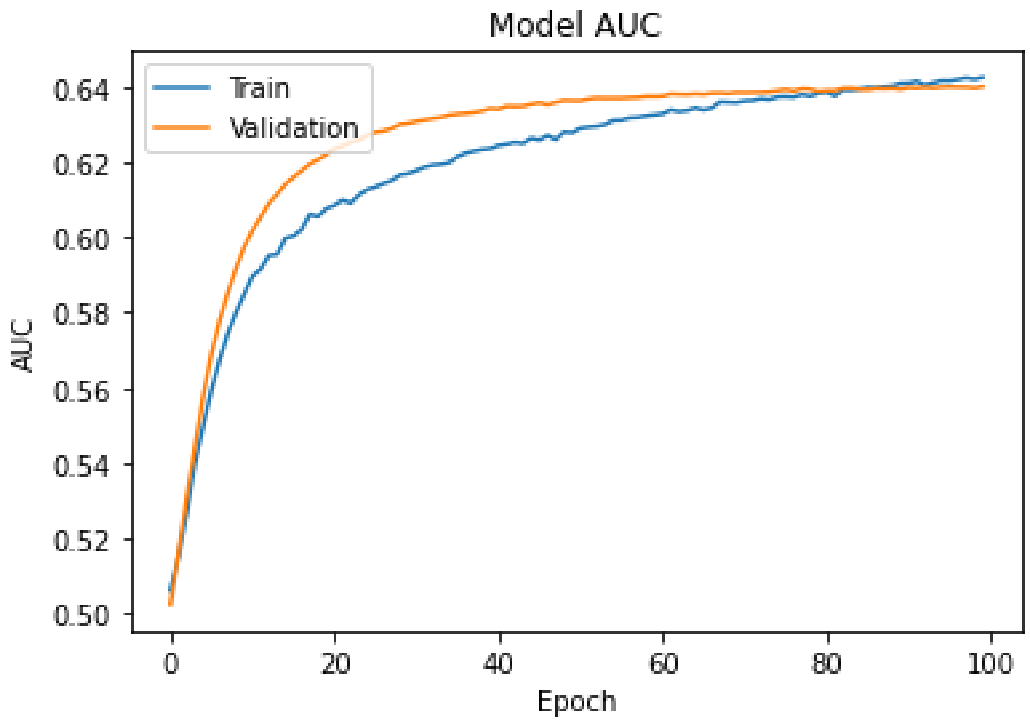
[20]:
```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



[21]:
```
# YOUR CODE HERE #
#print(hist.history)

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# END CODE #
```



[22]:
```
plt.plot(hist.history['auc_2'])
plt.plot(hist.history['val_auc_2'])
plt.title('Model AUC')
plt.ylabel('AUC')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



Q3.8 combining chunked predictions to full predictions for readmission

In the ClinicalBert paper, the authors did pretraining with MIMIC-III, and in section 3.3.2, they make predictions of hospital readmission on MIMIC-III, which is the same task we are doing.

We've broken our notes into chunks and made predictions for each chunk. The ClinicalBert authors propose a method to combine the chunked notes predictions into one prediction per note (equation 4 in the paper).

Implement this function in the next cell for the *validation* set with `c=1`. Use the combined predictions to compute the AUC.

[23]:
```
import sklearn
def predict_FULL_note_readmission_clinicalBert(model,␣
↪bert_pool_embeddings_CHUNKS,
                                                idxs_CHUNKS, c=1):
    """
    Combine CHUNK predictions to FULL predictions using equation 4 of
        https://arxiv.org/pdf/1904.05342.pdf

    Args:
    model (tf.keras.Model): a trained prediction model.
    bert_embedding_CHUNK (np.array[float,flat]): chunked dataset of bert␣
↪embeddings
        for running prediction.
    idxs_CHUNKS np.array([int]): idxs_CHUNKS[i]=j means chunk i is a␣
↪subsequence of
        a note i.

    Returns:
        y_pred_FULL (np.array([int]))
    """
    n_unique = len(np.unique(idxs_CHUNKS))
    idxs_CHUNKS = np.array(idxs_CHUNKS)
    y_pred_score_CHUNKS = model.predict(bert_pool_embeddings_CHUNKS)

    y_pred_FULL = np.zeros(n_unique)
    for i in range(n_unique):
        # YOUR CODE HERE #
        notes = np.where(idxs_CHUNKS == i)
        n = len(notes)
        chunks = y_pred_score_CHUNKS[notes]
        max_chunk = max(chunks)
        avg_chunks = np.average(chunks)
        readmit = (avg_chunks + (max_chunk[0] * (n/c))/(1 + (n/c)))
        y_pred_FULL[i] = readmit
        # END CODE #
    #y_pred_FULL = y_pred_FULL[y_pred_FULL != 0]
    #y_pred_FULL = sklearn.preprocessing.binarize(y_pred_FULL.reshape(-1, 1), 0.
↪5)
    return y_pred_FULL

c = 1
y_pred_FULL = predict_FULL_note_readmission_clinicalBert(model,
                                                                    ␣
↪val_bert_pool_embeddings_CHUNKS,
                                                                     val_idxs_CHUNKS,
                                                                     c=c)
y_score_FULL = val_labels_FULL
auc = None
# YOUR CODE HERE #
#auc = sklearn.metrics.roc_auc_score(y_pred_FULL, y_score_FULL)
auc = sklearn.metrics.roc_auc_score(y_score_FULL, y_pred_FULL)
#sklearn.metrics.roc_auc_score(y_true, y_score
# END CODE #
print(f"AUC {auc:.5f}")
```

```
AUC 0.69830
```

Q3.9 hyperparameter tuning

Run `predict_FULL_note_readmission_clinicalBert` and compute the AUC for a range of `c` values. We will use the best AUC to choose a value of `c`. This is hyperparameter tuning, and so we should do this on the validation set.

```
[24]: for c in [0.01,0.1,0.5,1,2,5,10,20,50]:
          auc = None
          # YOUR CODE HERE #
          y_pred_FULL = predict_FULL_note_readmission_clinicalBert(model,
                                                                           ␣
       ↪val_bert_pool_embeddings_CHUNKS,
                                                                            val_idxs_CHUNKS,
                                                                            c=c)
          auc = sklearn.metrics.roc_auc_score(y_score_FULL, y_pred_FULL)
          # END CODE #
          print(f"c {c}\t AUC: {auc:.5f}")
```

```
c 0.01   AUC: 0.69676
c 0.1    AUC: 0.69693
c 0.5    AUC: 0.69799
c 1      AUC: 0.69830
c 2      AUC: 0.69836
c 5      AUC: 0.69789
c 10     AUC: 0.69756
c 20     AUC: 0.69694
c 50     AUC: 0.69648
```

Q3.10 evaluation

Now that you have chosen a c value, let's evaluate on the test set. Run `predict_FULL_note_readmission_clinicalBert` and compute the AUC. In the next cell you just have to fill in your value of `c` and compute the `auc`.

```
[25]: c = 2 # choose your best value

      y_pred_FULL = predict_FULL_note_readmission_clinicalBert(model,
                                                                        ␣
       ↪test_bert_pool_embeddings_CHUNKS,
                                                                 test_idxs_CHUNKS,
                                                                 c=c)
      y_score_FULL = test_labels_FULL
      auc = None
      # YOUR CODE HERE #
      auc = sklearn.metrics.roc_auc_score(y_score_FULL, y_pred_FULL)
      # END CODE #
      print(f"Test set auc {auc:.5f}")
```

```
Test set auc 0.68074
```

Your test set AUC may be different to tha validation set. Explain why, and give one strategy for getting more consistent results between validation and test.

Written answer: randomly generated test set was easier to predict than the validation as the model was pre-trained on words that were more relevant to the test set. try and normalize btwn the sets.

we should do this on the validation set.

[24]:
```
for c in [0.01,0.1,0.5,1,2,5,10,20,50]:
    auc = None
    # YOUR CODE HERE #
    y_pred_FULL = predict_FULL_note_readmission_clinicalBert(model,
                                                             ␣
 ↪val_bert_pool_embeddings_CHUNKS,
                                                              val_idxs_CHUNKS,
                                                              c=c)
    auc = sklearn.metrics.roc_auc_score(y_score_FULL, y_pred_FULL)
    # END CODE #
    print(f"c {c}\t AUC: {auc:.5f}")
```

```
c 0.01   AUC: 0.69676
c 0.1    AUC: 0.69693
c 0.5    AUC: 0.69799
c 1      AUC: 0.69830
c 2      AUC: 0.69836
c 5      AUC: 0.69789
c 10     AUC: 0.69756
c 20     AUC: 0.69694
c 50     AUC: 0.69648
```

Q3.10 evaluation

Now that you have chosen a `c` value, let's evaluate on the test set. Run `predict_FULL_note_readmission_clinicalBert` and compute the AUC. In the next cell you just have to fill in your value of `c` and compute the `auc`.

[25]:
```
c = 2 # choose your best value

y_pred_FULL = predict_FULL_note_readmission_clinicalBert(model,
                                                         ␣
 ↪test_bert_pool_embeddings_CHUNKS,
                                                          test_idxs_CHUNKS,
                                                          c=c)
y_score_FULL = test_labels_FULL
auc = None
# YOUR CODE HERE #
auc = sklearn.metrics.roc_auc_score(y_score_FULL, y_pred_FULL)
# END CODE #
print(f"Test set auc {auc:.5f}")
```

```
Test set auc 0.68074
```

Your test set AUC may be different to tha validation set. Explain why, and give one strategy for getting more consistent results between validation and test.

Written answer: The randomly generated test set was more difficult to predict than the validation as the model was pre-trained on words that were more relevant to the validation set. A suitable. fix to this issue to get more consistent results between validation and test is to try and normalize between both datasets and ensure that words in the validation and test set notes chunks had relatively the same frequencies. As a result, we would see less imbalance between the test and validation set and see more consistent results.

```
[ ]:
```