Prediction of ICU Readmissions

Unplanned readmissions to ICU contribute to high health care costs and poor patient outcomes. 6-7% of all ICU cases see a readmission within 72 hours. Machine learning models on electronic health record data can help identify these cases, providing more information about short and long-term risks to clinicians at the time of ICU discharge

1. Topic

In this project I'll try to explore how to predict ICU readmission based on publicly available dataset (MIMIC) and optimize the model to get a reasonable performance

2. Dataset

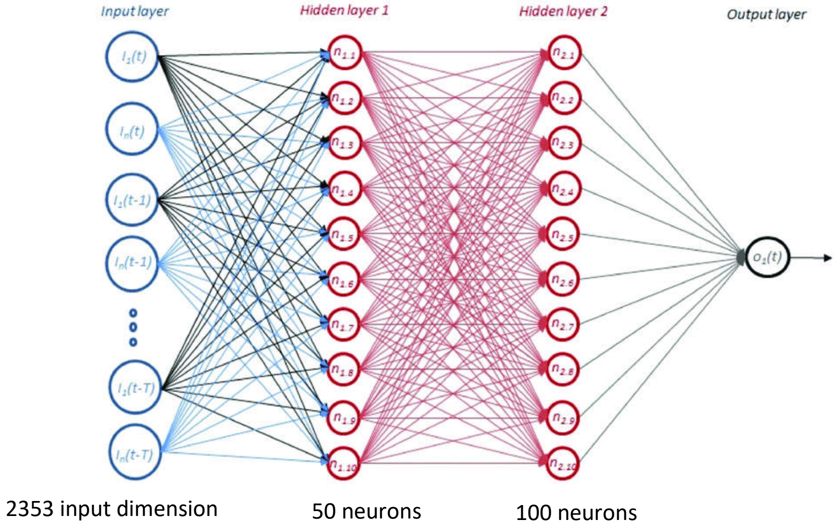
In MIMIC III dataset, (https://mimic.physionet.org/), we have a registry developed by the MIT Lab for Computational Physiology, comprising deidentified health data associated with ~40,000 critical care patients. It includes demographics, vital signs, laboratory tests, medications, and more.

In my github page (https://github.com/ZepengHuo/deep_learning_class.git), you can find the extracted data, stored in csv file format, named as df_MASTER_DATA.csv. It has 2354 features, ranging from heart rate, blood pressure, etc.

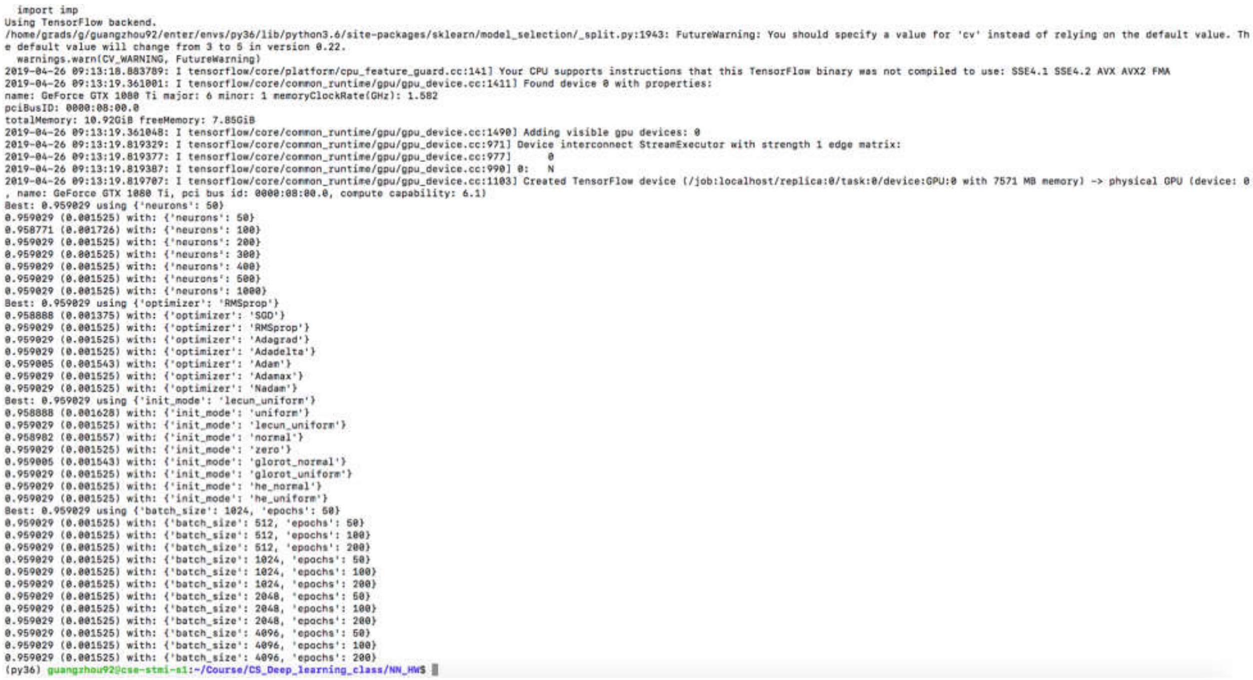
In the dataset, you can find out whether a patient has been readmitted in 72 hours, and I'll use that as my training labels

3. Neural network model

I used grid search to search all the optimal hyper-parameters (e.g. number of neurons) to train a neural network, code is (GridSearch_NN.py), result is (Grid_search_results.png).



As shown in the figure, I have input variables from the left and one output (readmission or not) on the right.



```
  import imp
Using TensorFlow backend.
/home/grads/g/guangzhou92/enter/envs/py36/lib/python3.6/site-packages/sklearn/model_selection/_split.py:1943: FutureWarning: You should specify a value for 'cv' instead of relying on the default value. Th
e default value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
2019-04-26 09:13:18.883789: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
2019-04-26 09:13:19.361001: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1411] Found device 0 with properties:
name: GeForce GTX 1080 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.582
pciBusID: 0000:08:00.0
totalMemory: 10.92GiB freeMemory: 7.85GiB
2019-04-26 09:13:19.361048: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1490] Adding visible gpu devices: 0
2019-04-26 09:13:19.819329: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-04-26 09:13:19.819377: I tensorflow/core/common_runtime/gpu/gpu_device.cc:977]      0
2019-04-26 09:13:19.819387: I tensorflow/core/common_runtime/gpu/gpu_device.cc:990] 0:   N
2019-04-26 09:13:19.819707: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1103] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 7571 MB memory) -> physical GPU (device: 0
, name: GeForce GTX 1080 Ti, pci bus id: 0000:08:00.0, compute capability: 6.1)
Best: 0.959029 using {'neurons': 50}
0.959029 (0.001525) with: {'neurons': 50}
0.958771 (0.001726) with: {'neurons': 100}
0.959029 (0.001525) with: {'neurons': 200}
0.959029 (0.001525) with: {'neurons': 300}
0.959029 (0.001525) with: {'neurons': 400}
0.959029 (0.001525) with: {'neurons': 500}
0.959029 (0.001525) with: {'neurons': 1000}
Best: 0.959029 using {'optimizer': 'RMSprop'}
0.958888 (0.001375) with: {'optimizer': 'SGD'}
0.959029 (0.001525) with: {'optimizer': 'RMSprop'}
0.959029 (0.001525) with: {'optimizer': 'Adagrad'}
0.959029 (0.001525) with: {'optimizer': 'Adadelta'}
0.959005 (0.001543) with: {'optimizer': 'Adam'}
0.959029 (0.001525) with: {'optimizer': 'Adamax'}
0.959029 (0.001525) with: {'optimizer': 'Nadam'}
Best: 0.959029 using {'init_mode': 'lecun_uniform'}
0.958888 (0.001628) with: {'init_mode': 'uniform'}
0.959029 (0.001525) with: {'init_mode': 'lecun_uniform'}
0.958982 (0.001557) with: {'init_mode': 'normal'}
0.959029 (0.001525) with: {'init_mode': 'zero'}
0.959005 (0.001543) with: {'init_mode': 'glorot_normal'}
0.959029 (0.001525) with: {'init_mode': 'glorot_uniform'}
0.959029 (0.001525) with: {'init_mode': 'he_normal'}
0.959029 (0.001525) with: {'init_mode': 'he_uniform'}
Best: 0.959029 using {'batch_size': 1024, 'epochs': 50}
0.959029 (0.001525) with: {'batch_size': 512, 'epochs': 50}
0.959029 (0.001525) with: {'batch_size': 512, 'epochs': 100}
0.959029 (0.001525) with: {'batch_size': 512, 'epochs': 200}
0.959029 (0.001525) with: {'batch_size': 1024, 'epochs': 50}
0.959029 (0.001525) with: {'batch_size': 1024, 'epochs': 100}
0.959029 (0.001525) with: {'batch_size': 1024, 'epochs': 200}
0.959029 (0.001525) with: {'batch_size': 2048, 'epochs': 50}
0.959029 (0.001525) with: {'batch_size': 2048, 'epochs': 100}
0.959029 (0.001525) with: {'batch_size': 2048, 'epochs': 200}
0.959029 (0.001525) with: {'batch_size': 4096, 'epochs': 50}
0.959029 (0.001525) with: {'batch_size': 4096, 'epochs': 100}
0.959029 (0.001525) with: {'batch_size': 4096, 'epochs': 200}
(py36) guangzhou92@cse-stmi-s1:~/Course/CS_Deep_learning_class/NN_HW$
```

The grid search result is shown above. And I used the best of each hyper-parameter to train the neural network. The architecture is dense network

X_train is 42664 rows × 2353 columns

X_test is 10207 rows × 2353 columns

Input layer: 2353

Hidden layer 1:

- 50 neurons
- Shape of tensor: 2353 * 50

Hidden layer 2:

- 100 neurons
- Shape of tensor: 50 * 100

Loss function: 'binary_crossentropy'

Output layer: 1 neuron

4. Annotated code

```
import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
device_nb = 2
os.environ["CUDA_VISIBLE_DEVICES"]=str(device_nb)

import pandas as pd
import datetime
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
import tensorflow as tf
from keras import backend as K
import tarfile
import numpy as np
import _pickle as cPickle
import os
import wfdb
from datetime import datetime
from datetime import timedelta
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.constraints import maxnorm

from sklearn.model_selection import KFold
import numpy as np
from sklearn.metrics import roc_auc_score
from sklearn.metrics import make_scorer



df_o = pd.read_csv("df_MASTER_DATA.csv")

import random
random.seed(9)
random.sample(range(1, 61089), 10000)

#sub-select 20% for testing
test_idx = random.sample(range(df_o.shape[0]), int(df_o.shape[0] * 0.2))
df_test = df_o.loc[test_idx]
train_idx = list(set([ x for x in range(df_o.shape[0])]) - set(test_idx))
df_train = df_o.loc[train_idx]
```

```
best_neurons = 50
best_optimizer = 'RMSprop'
best_init_mode = 'lecun_uniform'
best_batch_size =  1024
best_epochs = 50


def create_model(neurons=best_neurons, optimizer=best_optimizer, init_mode=best_init_mode,
batch_size=best_batch_size,
             epochs=best_epochs):
   # create model
   model = Sequential()
   model.add(Dense(neurons, input_dim=2291, activation='relu',
               kernel_initializer=init_mode, kernel_constraint=maxnorm(4)))
   model.add(Dropout(0.2))
   model.add(Dense(1, kernel_initializer=init_mode, activation='sigmoid'))
   # Compile model
   model.compile(loss='binary_crossentropy', optimizer=best_optimizer, metrics=['accuracy'])
   #model.compile(loss='binary_crossentropy', optimizer='adam')
   return model


#mean impute
#delete HADM_ID, SUBJECT_ID, ETHNICITY, MARITAL_STATUS, INSURANCE, RELIGION, INTIME, OUTTIME
#dummify
def preprocessing(df_labeled):


   #pre-processing, missingness > 0.5

   df_labeled_filtered = df_labeled

   for column in df_labeled_filtered.columns:
      missing_r = df_labeled[column].isnull().sum()/df_labeled.shape[0]
      if missing_r > 0.5:

         #keep arterial BP
         if not (column.startswith('Arterial_BP') or column.startswith('Mean_Arterial')):


            df_labeled_filtered = df_labeled_filtered.drop(columns=[column])


   #mean impute
   df_labeled_filtered = df_labeled_filtered.fillna(df_labeled_filtered.mean())

   #delete HADM_ID, SUBJECT_ID, ETHNICITY, MARITAL_STATUS, INSURANCE, RELIGION, INTIME, OUTTIME
   colmuns_todrop = ['Unnamed: 0','FIRST_CAREUNIT','ETHNICITY','MARITAL_STATUS','HADM_ID', 'ICUSTAY_ID',
               'INSURANCE','RELIGION','INTIME', 'OUTTIME', 'SUBJECT_ID','LANGUAGE','Time_To_readmission',
               'IsReadmitted_24hrs', 'IsReadmitted_48hrs',  'IsReadmitted_7days',
               'IsReadmitted_30days', 'IsReadmitted_Bounceback']
   #don't delete ==> 'IsReadmitted_72hrs',

   for column_d in colmuns_todrop:

      try:
         df_labeled_filtered = df_labeled_filtered.drop(columns=column_d)
```

```
        except:
            pass

    #dummify
    df_labeled_filtered = pd.get_dummies(df_labeled_filtered)


    return df_labeled_filtered




def fivefold_auroc(df_labeled_filtered):

    #Up- and down-sample for imbalance class
    X_1 = df_labeled_filtered[df_labeled_filtered['IsReadmitted_72hrs'] == 1]
    X_0 = df_labeled_filtered[df_labeled_filtered['IsReadmitted_72hrs'] == 0].iloc[:X_1.shape[0], :]

    X_all = X_0.append(X_1)

    X_all = df_labeled_filtered

    y = X_all['IsReadmitted_72hrs']
    X = X_all.drop(columns=['IsReadmitted_72hrs'])


    #X = df_labeled_filtered.drop(columns=['label'])
    #y = df_labeled_filtered['label']

    kf = KFold(n_splits=5, shuffle=True)
    kf.get_n_splits(X)

    fold_num = 1

    for train_index, test_index in kf.split(X):
        #print("TRAIN:", train_index, "TEST:", test_index)

        model = KerasClassifier(build_fn=create_model, epochs=best_epochs, batch_size=best_batch_size, verbose=0)



        model.fit(X.iloc[train_index], y.iloc[train_index])

        print('Fold %d' %fold_num, end=' ')
        fold_num += 1

        print(roc_auc_score(y.iloc[test_index], model.predict_proba(X.iloc[test_index])[:,1]))


df_filtered = preprocessing(df_train)
fivefold_auroc(df_filtered)
```

I have the code to run a 5 fold cross validation, and iterate through each of them, each time a 20% of data will be used as testing set, and the remaining will be used as training, until all data has been used.

5. Visualization/demonstration

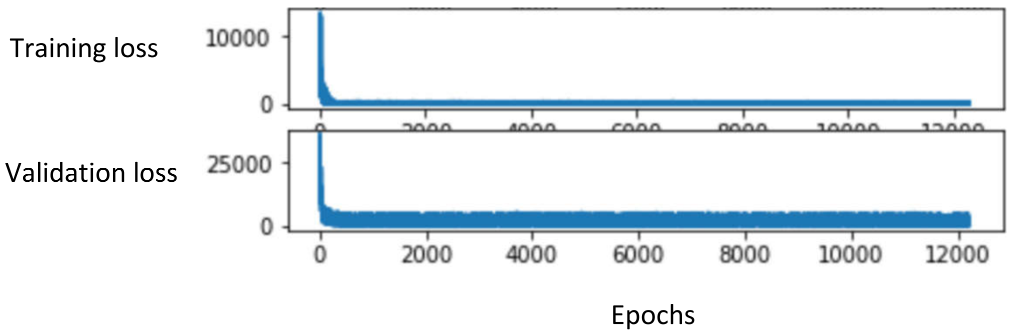
Run the commend in terminal:

```
python medical_prediction.py
```

And you will see input and output window. In my project, I'm planning to use medical data registry to predict whether the patient will be readmitted to an ICU (a minor revision from last video). Since readmission to an ICU in a short time will incur unnecessary overhead and potentially influence the mortality rate. So if the model can predict if a patient will be readmitted at what time, it can largely reduce the patient risk and increase survival rate

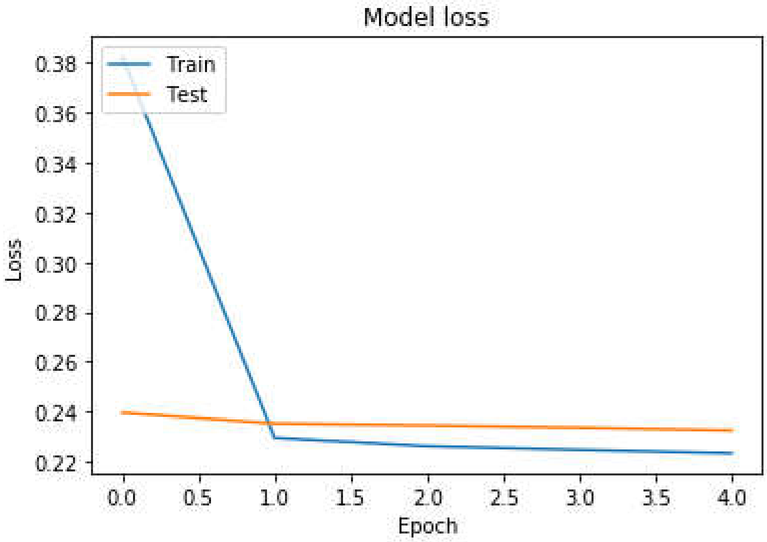
The demo will first require you generate test patient data from a test set (data here is a mockup), and then it will predict the readmission for those patients. Lastly it will plot the result of how well the model predicts.

Github link: https://github.com/ZepengHuo/deep_learning_class.git
Video link: https://youtu.be/KkjTmxNwnSs



For improvement, I've trained a LSTM model on only the time-series data in the MIMIC dataset

The loss look like this:



However, when I jointly train them together, by concatenation of LSTM to the dense network, the performance didn’t improve much, since the original accuracy is already really high (~0.96). So I concluded the dense network can already prediction readmission in an ideal performance. The code for LSTM implementation:

```
# coding: utf-8

# In[8]:


import pandas as pd
import datetime
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
import tarfile
import numpy as np
import _pickle as cPickle
import os
import wfdb
from datetime import datetime
from datetime import timedelta
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras import backend as K
```

```
# In[2]:


#df_CE = pd.read_csv('all_chart_events.csv')
#df_CE.columns = ['icustay_id', 'itemid', 'valuenum', 'charttime']


# In[3]:


#df_itemID = pd.read_csv('unique_chartevent_items.csv')
#df_itemID.columns = ['itemid', 'measurement']


# In[9]:


df_trainID = pd.read_csv('df_train_subjects.csv')
df_testID = pd.read_csv('df_test_subjects.csv')
df_valID = pd.read_csv('df_val_subjects.csv')


# In[10]:


df_trainID


# # bounce back has 6% of the data

# In[5]:


import os

path, dirs, files = next(os.walk("./each_icustay_csv"))
file_count = len(files)
file_count


# In[6]:


import os

path, dirs, files = next(os.walk("./feature_folder"))
file_count = len(files)
file_count
```

```
# # feature space has non-sparse ratio mean: 23%, max 51%, min 0%

# # get features prepared for LSTM (training set)

# In[22]:


#nan is from mean, std, that there's no mean, std (just one value in a window)
#0 is from max, min, count


x_train = []
y_train = []


for index, row in df_trainID.iterrows():

    icustay_id = row['ICUSTAY_ID']


    #see if a icu stay has a feature file
    try:
        outfile = './feature_folder/feature_space_%s.file.npy'%(str(icustay_id))
        all_feature_vectors = np.load(outfile)

        #change NaN to 0
        all_feature_vectors = np.nan_to_num(all_feature_vectors)


        #count non-zero
        #non_0 = np.count_nonzero(np.nan_to_num(all_feature_vectors))
        #non_0_ratio = non_0/(all_feature_vectors.shape[0]*all_feature_vectors.shape[1])

        x_train.append(all_feature_vectors)

        y_train.append(row['IsReadmitted_Bounceback'])

    except FileNotFoundError:
        continue

x_train = np.asarray(x_train)
y_train = np.asarray(y_train)


# In[24]:
```

```
x_train.shape


# # get features prepared for LSTM (validation set)

# In[8]:


#nan is from mean, std, that there's no mean, std (just one value in a window)
#0 is from max, min, count


x_valid = []
y_valid = []


for index, row in df_valID.iterrows():

    icustay_id = row['ICUSTAY_ID']


    #see if a icu stay has a feature file
    try:
        outfile = './feature_folder/feature_space_%s.file.npy'%(str(icustay_id))
        all_feature_vectors = np.load(outfile)

        #change NaN to 0
        all_feature_vectors = np.nan_to_num(all_feature_vectors)


        #count non-zero
        #non_0 = np.count_nonzero(np.nan_to_num(all_feature_vectors))
        #non_0_ratio = non_0/(all_feature_vectors.shape[0]*all_feature_vectors.shape[1])

        x_valid.append(all_feature_vectors)

        y_valid.append(row['IsReadmitted_Bounceback'])

    except FileNotFoundError:
        continue

x_valid = np.asarray(x_valid)
y_valid = np.asarray(y_valid)


# # get features prepared for LSTM (testing set)
# In[9]:


#nan is from mean, std, that there's no mean, std (just one value in a window)
#0 is from max, min, count


x_test = []
y_test = []


for index, row in df_testID.iterrows():

    icustay_id = row['ICUSTAY_ID']


    #see if a icu stay has a feature file
    try:
        outfile = './feature_folder/feature_space_%s.file.npy'%(str(icustay_id))
        all_feature_vectors = np.load(outfile)

        #change NaN to 0
        all_feature_vectors = np.nan_to_num(all_feature_vectors)


        #count non-zero
        #non_0 = np.count_nonzero(np.nan_to_num(all_feature_vectors))
        #non_0_ratio = non_0/(all_feature_vectors.shape[0]*all_feature_vectors.shape[1])

        x_test.append(all_feature_vectors)

        y_test.append(row['IsReadmitted_Bounceback'])

    except FileNotFoundError:
        continue

x_test = np.asarray(x_test)
y_test = np.asarray(y_test)


# # test: not output along time dimension, but along neurons dimesnion

# In[17]:


import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
device_nb = '7'
```

```
os.environ["CUDA_VISIBLE_DEVICES"]=str(device_nb)



from keras.models import Model
from keras.layers import Input
from keras.layers import LSTM, Dense
from numpy import array
# define model
inputs1 = Input(shape=(3, 1))
#lstm1 = LSTM(2, return_state=True)(inputs1)
lstm1 = LSTM(10, name='lstm')(inputs1)
output = Dense(1)(lstm1)
model = Model(inputs=inputs1, outputs=output)
# define input data
data = array([[0.8, 0.9, 0.7],[0.1, 0.2, 0.3]]).reshape((2,3,1))
# make and show prediction
prediction=model.predict(data)
print(prediction)





# In[18]:


df = pd.DataFrame(data.reshape(2,-1))
df.values


# In[6]:


model.summary()


# In[20]:


layer_name = 'lstm'
intermediate_layer_model_output = Model(inputs=model.input,
                                outputs=model.get_layer(layer_name).output)
intermediate_output = intermediate_layer_model_output.predict(df.values.reshape(2,3,1))


# In[21]:
```

```
intermediate_output


# In[25]:


pd.DataFrame(intermediate_output)


# In[2]:


from numba import cuda
cuda.select_device(0)
cuda.close()


# In[26]:


cohortvector_train = pd.read_csv('cohortvector_train.csv')


# In[27]:


cohortvector_train


# # LSTM

# In[10]:


import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
device_nb = 2
os.environ["CUDA_VISIBLE_DEVICES"]=str(device_nb)



model = Sequential()
#model.add(LSTM(32, return_sequences=True, input_shape=(48, 30)))
model.add(LSTM(2048, input_shape=(48, 30)))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')
```

```
history = model.fit(x_train, y_train, epochs=5, batch_size=2014, verbose=2, validation_data=(x_valid,
y_valid))


y_pred_proba = model.predict_proba(x_test)
y_pred = model.predict(x_test)


# In[23]:


from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, y_pred_proba)


# In[12]:


import matplotlib.pyplot as plt
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()


# In[22]:


y_pred_proba


# In[21]:


y_pred


# In[6]:


from numba import cuda
cuda.select_device(0)
cuda.close()
```

```
# In[15]:


itemid1 = 211
itemid2 = 224697

icustay_id = 294638


intime = pd.to_datetime(df_trainID[df_trainID['ICUSTAY_ID'] == icustay_id]['INTIME'].values[0])

outtime = pd.to_datetime(df_trainID[df_trainID['ICUSTAY_ID'] == icustay_id]['OUTTIME'].values[0])



with open("./each_icustay_csv/" + str(icustay_id) + '.csv') as f0:
    Adf_ICUstay_CE = pd.read_csv(f0)


onetime = pd.to_datetime(Adf_ICUstay_CE[(Adf_ICUstay_CE['itemid'] == itemid1) |
(Adf_ICUstay_CE['itemid'] == itemid2)]['charttime']).values[0]
df_onechannel = Adf_ICUstay_CE[(Adf_ICUstay_CE['itemid'] == itemid1) | (Adf_ICUstay_CE['itemid']
== itemid2)]

#Adf_ICUstay_CE[(Adf_ICUstay_CE['itemid'] == itemid1) | (Adf_ICUstay_CE['itemid'] == itemid2)]




# In[16]:


pd.to_datetime(df_onechannel['charttime'])
df_onechannel['charttime'] = pd.to_datetime(df_onechannel['charttime'])
df_onechannel['charttime'] .values[0]


# In[17]:


#given a icu stay's df, and what itemid, to get features in 48h

def OnefeaturesIN48h(Adf_ICUstay_CE, itemid1, itemid2, intime, outtime):

    feature_vector = []
    df_onechannel = Adf_ICUstay_CE[(Adf_ICUstay_CE['itemid'] == itemid1) | (Adf_ICUstay_CE['itemid']
== itemid2)]

    #change 'charttime' str to datetime object
    df_onechannel['charttime'] = pd.to_datetime(df_onechannel['charttime'])

    #TO DO:  first impute all of them

    end_time = intime

    #first 24 hours
    for pasthours in range(0, 24):


        start_time = end_time
        end_time = start_time + np.timedelta64(1, 'h')

        df_InATimewindow = df_onechannel[ (start_time < df_onechannel['charttime']) &
(df_onechannel['charttime'] < end_time) ]


        #add mean feature
        try:
            mean_feature = np.mean(df_InATimewindow['valuenum'].values)
        except ValueError:
            mean_feature = 0


        #add std feature
        try:
            std_feature = np.std(df_InATimewindow['valuenum'].values)
        except ValueError:
            std_feature = 0

        #add max feature
        try:
            max_feature = np.amax(df_InATimewindow['valuenum'].values)
        except ValueError:
            max_feature = 0

        #add min feature
        try:
            min_feature = np.amin(df_InATimewindow['valuenum'].values)
        except ValueError:
            min_feature = 0

        #add count feature
        try:
            count_feature = df_InATimewindow['valuenum'].values.shape[0]
        except ValueError:
            count_feature = 0

        features_1h_window = [mean_feature, std_feature, max_feature, min_feature, count_feature]

        feature_vector.append(features_1h_window)



    end_time = outtime - np.timedelta64(24, 'h')

    #last 24 hours
    for pasthours in range(0, 24):

        start_time = end_time
        end_time = start_time + np.timedelta64(1, 'h')

        df_InATimewindow = df_onechannel[ (start_time < df_onechannel['charttime']) &
(df_onechannel['charttime'] < end_time) ]


        #add mean feature
        try:
            mean_feature = np.mean(df_InATimewindow['valuenum'].values)
        except ValueError:
            mean_feature = 0


        #add std feature
        try:
            std_feature = np.std(df_InATimewindow['valuenum'].values)
        except ValueError:
            std_feature = 0

        #add max feature
        try:
            max_feature = np.amax(df_InATimewindow['valuenum'].values)
        except ValueError:
            max_feature = 0

        #add min feature
        try:
            min_feature = np.amin(df_InATimewindow['valuenum'].values)
        except ValueError:
            min_feature = 0

        #add count feature
        try:
            count_feature = df_InATimewindow['valuenum'].values.shape[0]
        except ValueError:
            count_feature = 0


        features_1h_window = [mean_feature, std_feature, max_feature, min_feature, count_feature]

        feature_vector.append(features_1h_window)



    return feature_vector


# In[18]:


def AICUstay_goodchannels(df_CE, df_itemID, icustay_id):
    goodchannels = []
    for item_count in df_CE[df_CE['icustay_id'] == icustay_id]['itemid'].value_counts().iteritems():
        itemid = item_count[0]
        item_name = df_itemID[df_itemID['itemid']==itemid]['measurement'].values[0]
        goodratio =  missing_ratio(df_CE, icustay_id, itemid)
        if goodratio >= 0.5:
            goodchannels.append(item_name)
            #print(item_name, goodratio)
    return goodchannels


# In[19]:


def missing_ratio(df_CE, icustay_id, itemid):
    return (df_CE[(df_CE['icustay_id'] == icustay_id) & (df_CE['itemid'] ==
itemid)]['valuenum'].shape[0]-df_CE[(df_CE['icustay_id'] == icustay_id) & (df_CE['itemid'] ==
itemid)]['valuenum'].isnull().sum()) / df_CE[(df_CE['icustay_id'] == icustay_id) & (df_CE['itemid'] ==
itemid)]['valuenum'].shape[0]
```