CS598 DL4H Project Report Spring 2023

Sagar Dalwadi and Murtaza Lodgher
{sagardd2, lodgher2}@illinois.edu
Group ID: 83
Paper ID: 12

Presentation link: https://youtu.be/N62x_KYRfrQ
Code link: https://github.com/malodgher/dlh-project-sp23.git

1. Introduction

In the medical field, heart failure (HF) is a difficult disease to detect early on and diagnose to a patient as many of the early signs and symptoms are common across different kinds of common diseases. As a result, the "onset of HF is associated with a high level of disability, health care costs, and mortality" (Choi et al., 2017, pg. 362). A paper by Edward Choi, Andy Schuetz, Walter F Stewart and Jimeng Sun, published in the *Journal of the American Medical Informatics Association* explores how Recurrent neural network (RNN) models can be used to detect HF as early as possible, even more so than other deep learning methods, as they are tuned to temporal relations. Our team will attempt to reproduce the results from this paper and add extra parameters to see how the model will react to different circumstances.

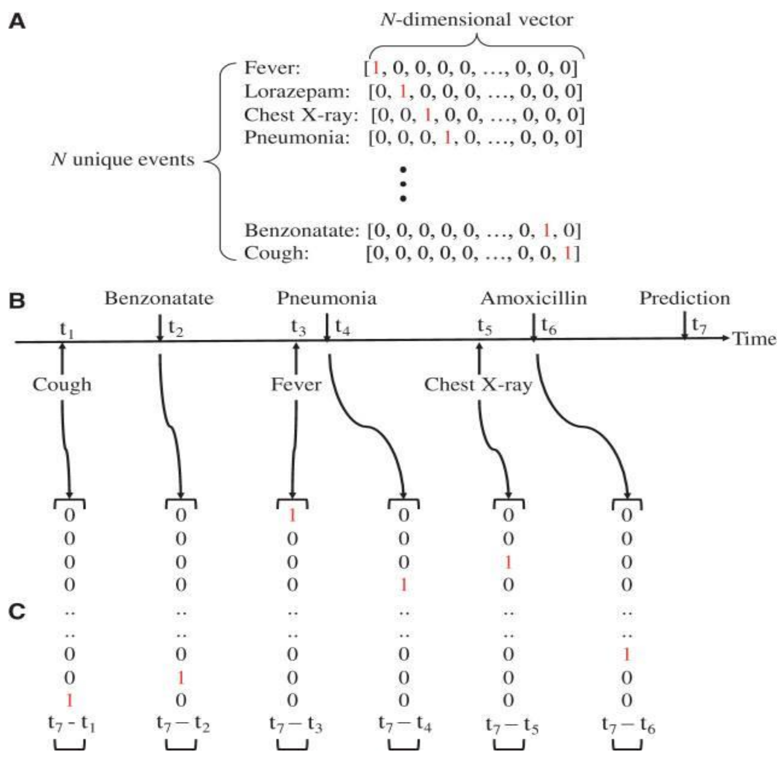
2. Scope of Reproducibility

The claims from this paper we have chosen to reproduce are as follows:

- RNN models using gated recurrent units (GRUs) were able to yield a higher area-under-the-curve (AUC) for predicting HF diagnoses through electronic health record (EHR) data over conventional deep learning methods, like logistic regression, SVM and KNN.

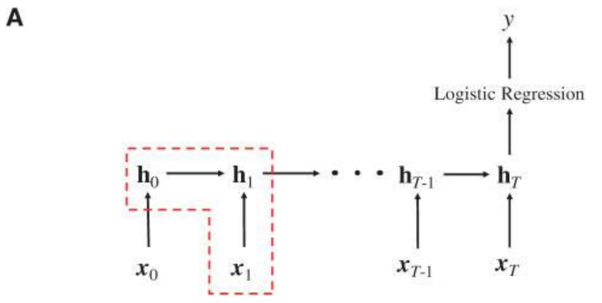
3. Methodology

3.1. Model Descriptions

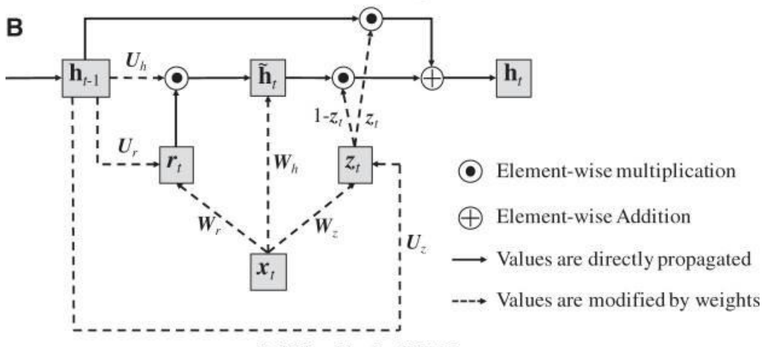

A
N-dimensional vector
Fever: [1, 0, 0, 0, 0, …, 0, 0, 0]
Lorazepam: [0, 1, 0, 0, 0, …, 0, 0, 0]
Chest X-ray: [0, 0, 1, 0, 0, …, 0, 0, 0]
Pneumonia: [0, 0, 0, 1, 0, …, 0, 0, 0]
N unique events
Benzonatate: [0, 0, 0, 0, 0, …, 0, 1, 0]
Cough: [0, 0, 0, 0, 0, …, 0, 0, 1]
B
Benzonatate
Pneumonia
Amoxicillin
Prediction
Time
Cough
Fever
Chest X-ray
C
t7 - t1
t7 − t2
t7 − t3
t7 − t4
t7 − t5
t7 − t6

To construct the GRU model, we first take the sequence of symptoms and convert them into hot-vectors using labels ranging from 0 - 1 to alter the dimension. We then arrange the vectors in order based on the time each symptom occurred, before a prediction was made. This concept is demonstrated in the image taken directly from the paper, where section A is converting the symptoms to hot-vectors, section B arranges the vectors based on time, assuming a prediction is made at time t7, and section C appends the time feature at the end of each hot vector:

This vector is then used as the input vector x_t for the GRU, where t is timestep for the number of clinical visits T. The vector is transformed and stored into a hidden layer $\boldsymbol{h}$, whose state changes over time as more input vectors are added to $\boldsymbol{h}$ in each timestep. After the final input vector x_T has been added to $\boldsymbol{h}$, a logistic regression function is applied and it produces the scalar vector y, representing the estimated patient-specific score for future diagnosis of HF. The model structure is illustrated in section A of the image below:



Model architecture for heart failure prediction with GRU


B
ht-1
Uh
h̃t
ht
1-zt
zt
Ur
rt
Wh
Wr
Wz
Uz
xt
Element-wise multiplication
Element-wise Addition
Values are directly propagated
Values are modified by weights

Building block of GRU

As shown in Section B of the image, The GRU model has four components: z_t: the update gate at time step t, r_t: the reset gate at time step t, $\sim h_t$: the intermediate memory unit at time step t and h_t: the hidden layer at time step t.

Mathematical formulation of GRU components:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$\sim h_t = tanh\,(W_h x_t + r_t \odot U_h h_{t-1} + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t\,) \odot \sim h_t$$

The logistic regression model was applied to the final state of the hidden layer as formulated below:

$$y = \sigma\,(w^T h_T + b)$$

And the loss function used for minimization is:

$$\text{Loss} = -\sum_{i=1}^{P} (c^{(i)} log\, y^{(i)} + (1 - c^{(i)}) log\, (1 - y^{(i)})\,)$$

3.2. Data Descriptions

Initially we ran our implementation based on the synthetic data provided by authors and later we switched to the real word data as described below.

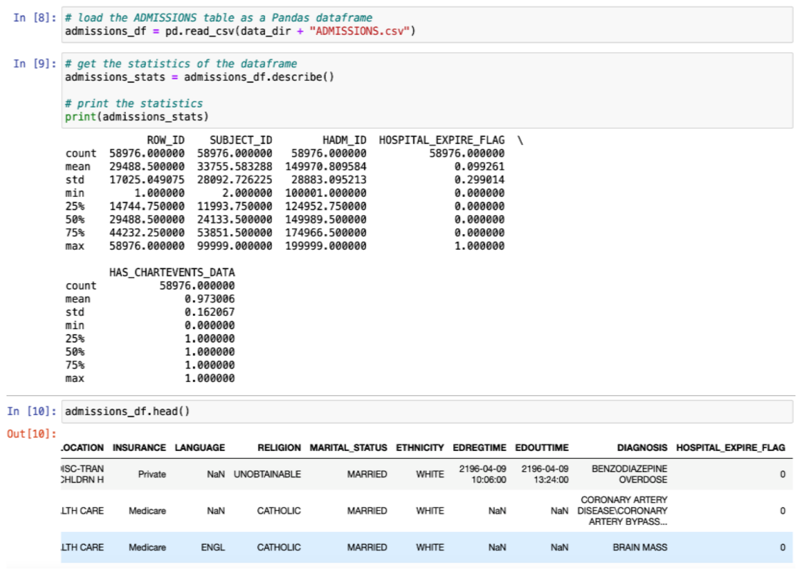
The datasets are extracted from PhysioNet website: (you need to have "CITI Data or Specimens Only Research" training completed & approved before you can access the data): https://physionet.org/content/mimiciii/1.4/

This Data includes following:

1. Admissions: gives information regarding a patient's admission to the hospital
2. Patients: Provides information on each patient
3. Diagnoses ICD: Contains ICD diagnoses for patients
4. ICU Stays: defines a single ICU stay
5. Lab events: Contains all laboratory measurements for a given patient

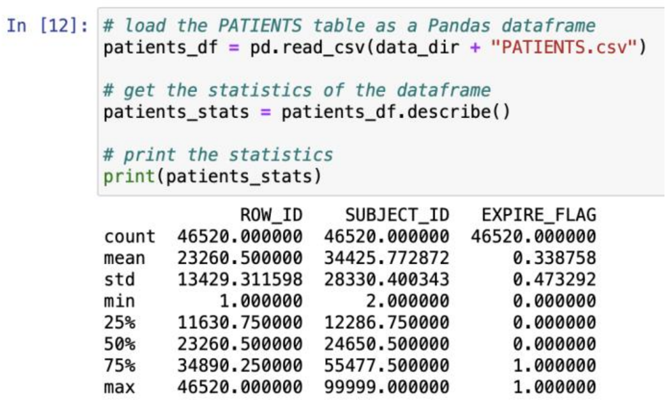
More information on dataset, fields etc can be found at: https://mimic.mit.edu/docs/iii/tables/

The ICD diagnosis code data was essential in deriving the target variable of heart failure indicator. The Patients data, diagnoses data as well as the lab events for patients were critical in initializing the One hot vector, building and running data through the GRU for HF prediction. Here are some statistics of the data we used (more details can be seen here). The statistics of the hospital admissions data, the statistics of the patient data and the lab events data are shown:

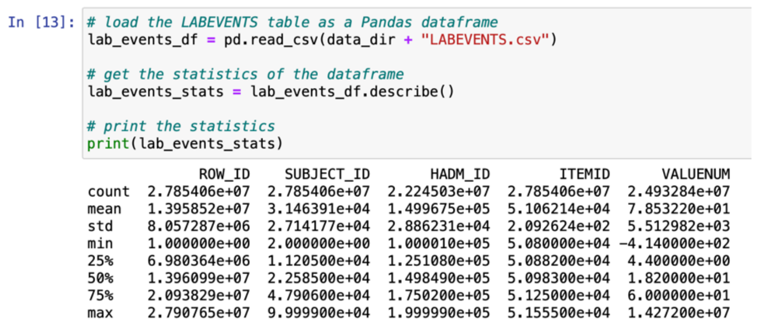


```
In [8]: # load the ADMISSIONS table as a Pandas dataframe
        admissions_df = pd.read_csv(data_dir + "ADMISSIONS.csv")

In [9]: # get the statistics of the dataframe
        admissions_stats = admissions_df.describe()

        # print the statistics
        print(admissions_stats)

             ROW_ID    SUBJECT_ID        HADM_ID  HOSPITAL_EXPIRE_FLAG  \
count  58976.000000  58976.000000   58976.000000          58976.000000
mean   29488.500000  33755.583288  149970.809584              0.099261
std    17025.049075  28092.726225   28883.095213              0.299014
min        1.000000      2.000000  100001.000000              0.000000
25%    14744.750000  11993.750000  124952.750000              0.000000
50%    29488.500000  24133.500000  149989.500000              0.000000
75%    44232.250000  53851.500000  174966.500000              0.000000
max    58976.000000  99999.000000  199999.000000              1.000000

       HAS_CHARTEVENTS_DATA
count          58976.000000
mean               0.973006
std                0.162067
min                0.000000
25%                1.000000
50%                1.000000
75%                1.000000
max                1.000000

In [10]: admissions_df.head()
Out[10]:
```

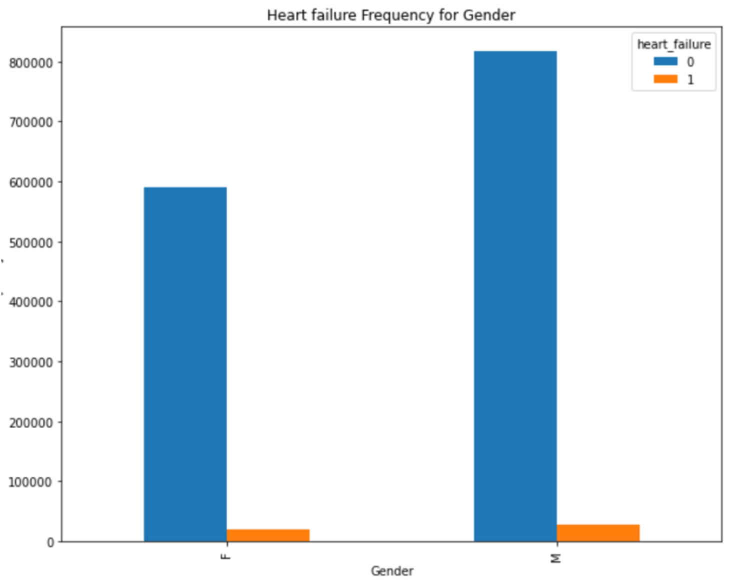
.OCATION	INSURANCE	LANGUAGE	RELIGION	MARITAL_STATUS	ETHNICITY	EDREGTIME	EDOUTTIME	DIAGNOSIS	HOSPITAL_EXPIRE_FLAG
ISC-TRAN CHLDRN H	Private	NaN	UNOBTAINABLE	MARRIED	WHITE	2196-04-09 10:06:00	2196-04-09 13:24:00	BENZODIAZEPINE OVERDOSE	0
LTH CARE	Medicare	NaN	CATHOLIC	MARRIED	WHITE	NaN	NaN	CORONARY ARTERY DISEASE\CORONARY ARTERY BYPASS...	0
LTH CARE	Medicare	ENGL	CATHOLIC	MARRIED	WHITE	NaN	NaN	BRAIN MASS	0



```
In [12]: # load the PATIENTS table as a Pandas dataframe
         patients_df = pd.read_csv(data_dir + "PATIENTS.csv")

         # get the statistics of the dataframe
         patients_stats = patients_df.describe()

         # print the statistics
         print(patients_stats)

             ROW_ID    SUBJECT_ID  EXPIRE_FLAG
count  46520.000000  46520.000000  46520.000000
mean   23260.500000  34425.772872      0.338758
std    13429.311598  28330.400343      0.473292
min        1.000000      2.000000      0.000000
25%    11630.750000  12286.750000      0.000000
50%    23260.500000  24650.500000      0.000000
75%    34890.250000  55477.500000      1.000000
max    46520.000000  99999.000000      1.000000
```



```
In [13]: # load the LABEVENTS table as a Pandas dataframe
         lab_events_df = pd.read_csv(data_dir + "LABEVENTS.csv")

         # get the statistics of the dataframe
         lab_events_stats = lab_events_df.describe()

         # print the statistics
         print(lab_events_stats)

             ROW_ID    SUBJECT_ID       HADM_ID        ITEMID      VALUENUM
count  2.785406e+07  2.785406e+07  2.224503e+07  2.785406e+07  2.493284e+07
mean   1.395852e+07  3.146391e+04  1.499675e+05  5.106214e+04  7.853220e+01
std    8.057287e+06  2.714177e+04  2.886231e+04  2.092624e+02  5.512982e+03
min    1.000000e+00  2.000000e+00  1.000010e+05  5.080000e+04 -4.140000e+02
25%    6.980364e+06  1.120500e+04  1.251080e+05  5.088200e+04  4.400000e+00
50%    1.396099e+07  2.258500e+04  1.498490e+05  5.098300e+04  1.820000e+01
75%    2.093829e+07  4.790600e+04  1.750200e+05  5.125000e+04  6.000000e+01
max    2.790765e+07  9.999900e+04  1.999990e+05  5.155500e+04  1.427200e+07
```



After deriving HF indicator for patients, above bar chart was created where the resulting bar chart displays the frequency of heart failure cases for each gender, with each bar representing a different gender. This type of chart is useful for comparing the number of cases between different categories and identifying any differences or patterns between them.

We also created a set of histograms to visualize the distribution of the data for each variable in the input data frame. Charts can be seen here The resulting figure shows a set of subplots, with each subplot displaying a histogram for a different variable. This can help to provide insight into the distribution of the data, such as whether it is normally distributed or skewed, and identify any outliers or patterns in the data.

3.3. Hyperparameters

The following hyperparameter settings were used to train GRU model:

- Units: the number of units/neurons in the GRU layer. It is set to 32
- Activation: the activation function used in the GRU and Dense layers. The GRU layer uses the 'relu' activation function, while the Dense layers use the 'relu' and 'sigmoid' activation functions.
- Dropout: the dropout rate used in the Dropout layer. It is set to 0.2
- Optimizer: the optimization algorithm used for training the model. In this code, it is set to Adam optimizer with a learning rate of 0.001.
- Loss: the loss function used to compute the difference between the predicted and actual values. In this code, it is set to 'binary_crossentropy'.
- Epochs: the number of times the entire training dataset is passed through the neural network during training. It is set to 3 for testing

3.4. Implementation

As mentioned earlier in the draft, initially we utilized author's code along with synthetic data to be able to gain understanding of the model and later we implemented our own code with real world data. The code we used for our implementation, while going by the framework of the original code base (link is here), is entirely written by members of our team and it implements the ablations laid out in our project proposal as well as below:

- The first step of our implementation involves reading in the data files, describing the statistics for each data file, and merging them into one pandas data frame.
- This data frame was then converted into One hot vector encoding.
- We then implemented the GRU model using the keras library's Sequential class. Our ablation comes from the activation we used when initializing the GRU. For one GRU model, we set the activation function to ReLU and for another model, we set the activation function to Sigmoid. Our team wanted to see if different activation functions would lead to different results (drastic or otherwise) than the results given in the original paper.
- A model with ReLU: it defines a neural network model using Keras with three layers: an Embedding layer, a GRU layer, and two dense layers. The Embedding layer is used to map the input data to a lower-dimensional space, while the GRU layer is a type of recurrent neural network that can model sequential data. The two dense layers are fully connected layers that add nonlinearity to the model. The activation functions used in the model are ReLU for the GRU and the first dense layer, and Sigmoid for the final dense layer. A dropout layer is also included to help prevent overfitting
- Finally, we split the data in training & testing and ran through the GRU models and outputted the resulting accuracy.

3.5. Computational Requirements

We used the personal machine with 2.3 Ghz 8-Core, 16 GB 2400 MHz DDR4, UHD Graphics 630 specifications to execute our implementation. Initially, there were some computational issues with our code, with timeouts and such, and we were considering using an outside vendor like AWS to run our code on. However, we were able to run our code by modifying some of the parameters:

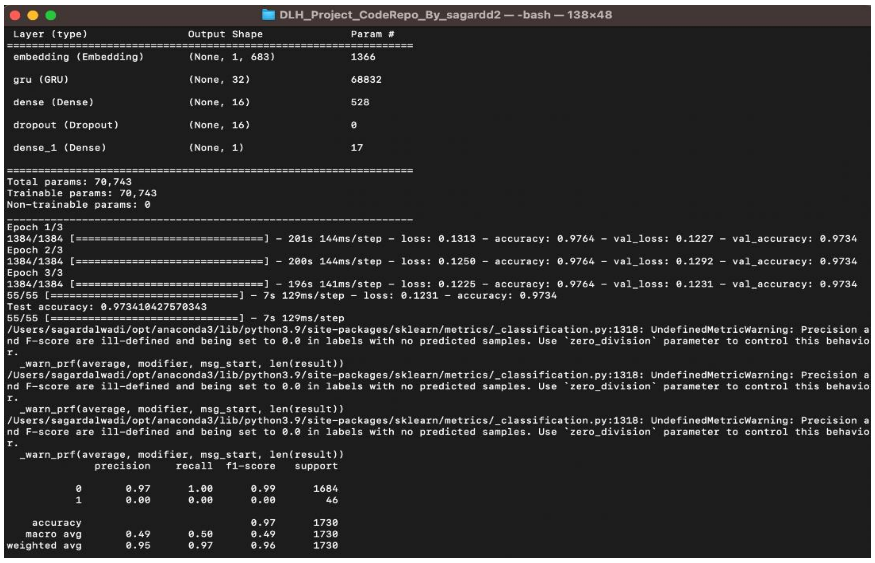
- Code with Sigmoid activation, 3 epochs, 1000 input records: To train 1 epoch, takes ~3 hours
- Code with ReLu activation, 3 epochs, 15000 input records: To train 1 epoch, takes ~5 min
- Code with ReLu activation, 3 epochs, full data (~700k): To train 1 epoch, takes ~9 hours

4. Results

The results that the code was able to output, regardless of our ablations, was able to match the main claim in our scope of reproducibility, that RNN models are able to yield a larger AUC for predicting HF than other traditional models.

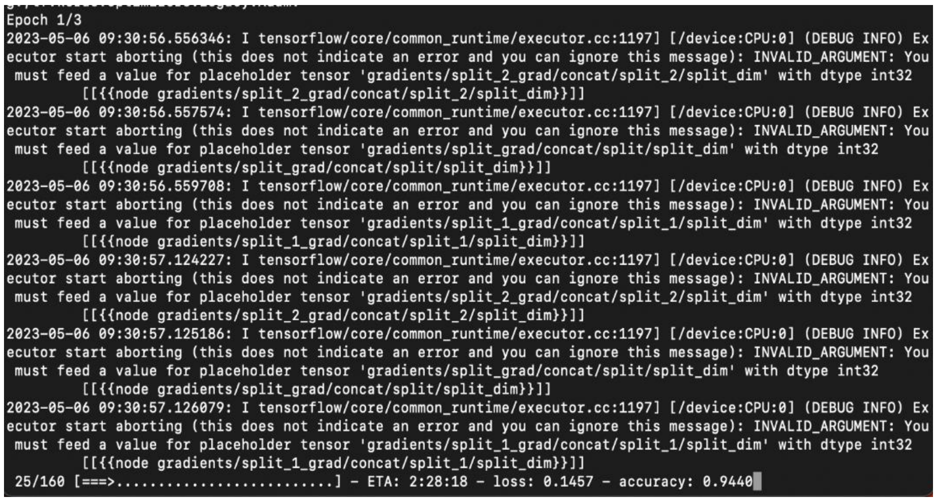
Result 1

The results from the image on the right are from a run of the data with the GRU initialized to a ReLU activation function. The accuracy of the model is outputted to be at 0.97, with a loss at about 0.12 – 0.13 per epoch. While the highest accuracy given from the paper with a one-hot vector utilization was about 0.75, this still represents a large factor of accuracy for using RNN for predictive analysis of HF. The difference in accuracy may be attributed to the difference in data used as well as input features available in data we used vs data authors used.



```
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 1, 683)            1366

 gru (GRU)                   (None, 32)                68832

 dense (Dense)               (None, 16)                528

 dropout (Dropout)           (None, 16)                0

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 70,743
Trainable params: 70,743
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
1384/1384 [==============================] - 201s 144ms/step - loss: 0.1313 - accuracy: 0.9764 - val_loss: 0.1227 - val_accuracy: 0.9734
Epoch 2/3
1384/1384 [==============================] - 200s 144ms/step - loss: 0.1250 - accuracy: 0.9764 - val_loss: 0.1292 - val_accuracy: 0.9734
Epoch 3/3
1384/1384 [==============================] - 196s 141ms/step - loss: 0.1225 - accuracy: 0.9764 - val_loss: 0.1231 - val_accuracy: 0.9734
55/55 [==============================] - 7s 129ms/step - loss: 0.1231 - accuracy: 0.9734
Test accuracy: 0.973410427570343
55/55 [==============================] - 7s 129ms/step
/Users/sagardalwadi/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/sagardalwadi/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/sagardalwadi/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           0       0.97      1.00      0.99      1684
           1       0.00      0.00      0.00        46

    accuracy                           0.97      1730
   macro avg       0.49      0.50      0.49      1730
weighted avg       0.95      0.97      0.96      1730
```

Result 2



```
Epoch 1/3
2023-05-06 09:30:56.556346: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Ex
ecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You
 must feed a value for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-06 09:30:56.557574: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Ex
ecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You
 must feed a value for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-06 09:30:56.559708: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Ex
ecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You
 must feed a value for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
2023-05-06 09:30:57.124227: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Ex
ecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You
 must feed a value for placeholder tensor 'gradients/split_2_grad/concat/split_2/split_dim' with dtype int32
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-05-06 09:30:57.125186: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Ex
ecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You
 must feed a value for placeholder tensor 'gradients/split_grad/concat/split/split_dim' with dtype int32
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-05-06 09:30:57.126079: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Ex
ecutor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You
 must feed a value for placeholder tensor 'gradients/split_1_grad/concat/split_1/split_dim' with dtype int32
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
 25/160 [===>..........................] - ETA: 2:28:18 - loss: 0.1457 - accuracy: 0.9440
```

The results from the image on the left are from a run of the data with the GRU initialized to a Sigmoid activation function. The accuracy of the model is outputted to be at 0.94, with a loss at about 0.14 – 0.15 per epoch. While a slightly lower accuracy than the ReLU activation function, this still supports the claim that, with any activation function, a RNN model will yield the highest accuracy than compared to conventional models.

5. Discussion

5.1. What was easy

One of the easier aspects for our team when developing or implementing was that, the GRU model description was quite detailed, and we were able to develop our model very similarly to what was described in the paper. Another benefit that our team had was that we had a code base written by the authors that our team could reference for any technical code implementations.

5.2. What was difficult

- One of the difficult aspects of creating our implementation was trying to find data that had a similar structure to the synthetic data given in the author's attached code repository. The actual data that the author's mentioned in the paper was Sutter Palo Alto Medical Foundation, which may have required some amount of payment to access the same data, so we had to go with data from Physio.Net that is free data from various medical institutions and we had to make sure that we could derive the same data structure as the synthetic data from this free data.
- Understanding of the data is very critical. Without understanding what data you are dealing with, it is very difficult to build any models or even identify important features.
- Another issue our team ran into was that we couldn't reproduce the number of epochs that were done in the original code due to limited hardware. As mentioned earlier, this problem was resolved by modifying the parameters in code somewhat, i.e., reducing the number of epochs generated when training.

5.3. Recommendations for reproducibility

- Make sure that one has powerful hardware to run the same number of epochs as the original code
- If possible, try to collect the data from the original source than through free resources to get the most optimum results
- Know that one doesn't need to use the exact same libraries as described in the original code. If there are libraries that can produce and train a model more efficiently, go ahead and use those.

6. Communication with the original authors

Our team did not have any communication with the original authors of the paper. Our implementation and findings were based solely on our team member's own interpretations of the paper.

7. Team members contribution statement

- The team members were in constant communication with each other and were able to update each other on their work status. Both team members had some part to play in making sure the materials needed to be created and submitted were done so accordingly.

- The code implementation was written by Sagar Dalwadi. Initially, our decision was to have both members work on the different ablations (with Sagar working on the ReLU code and Murtaza working on the Sigmoid code), however, Murtaza was running into many issues with his implementation, with sections not properly connected and needed to be rewritten entirely, so we decided to tweak Sagar's implementation to account for the Sigmoid ablation.

- Zoom meetings and the mobile group chat were facilitated by Murtaza Lodgher. Both the initial project draft and the final draft were written jointly. The presentation was also a joint effort between our two team members.

References

1. Choi, E., Schuetz, A., Stewart, W. F., & Sun, J. (2017). Using recurrent neural network models for early detection of heart failure onset. *Journal of the American Medical Informatics Association : JAMIA*, *24*(2), 361–370. https://doi.org/10.1093/jamia/ocw112

2. https://physionet.org/content/mimiciii/1.4/

3. https://mimic.mit.edu/docs/iii/tables/

4. https://github.com/mp2893/rnn_predict