# CS598 Project: Readmission prediction via deep contextual embedding of clinical concept

**Gowtham Kuntumalla and Yiming Li**
{gowtham4, yl159}@illinois.edu

Group ID: 100
Paper ID: 168
Presentation link: https://www.youtube.com/watch?v=tdRc92b4_EI
Code link: https://github.com/yl159-Yiming/CS598-DL4H

## 1 Introduction

We chose the paper titled 'Readmission prediction via deep contextual embedding of clinical concepts' (Xiao et al., 2018) for our academic paper reproduction project in the course CS598: Deep Learning for Healthcare, Spring 2023, UIUC.

This paper deals with the expensive problem of forecasting if a patient admitted to a hospital is likely to be readmitted again within 30 days of discharge. It is estimated that 17.6% of the patients discharged are readmitted to the hospital within 30 days. This translates to a monetary cost of $17.9 billion/year in the United States (Basu Roy et al., 2015).

Dealing with patients' Electronic Health Records (EHR) is difficult due to privacy issues, noisiness, heterogeneity, incompleteness, etc. Feature engineering required to solve these issues is nonintuitive and could be difficult to justify from both clinical and computational perspectives. Deep learning models are known for their end-to-end learning capabilities without an explicit need for feature engineering.

The authors in the paper (Xiao et al., 2018) propose a deep learning model called CONTENT. This model essentially predicts patient hospital readmissions using temporal data. It uses learning interpretable patient representations by capturing both local and global contexts from patients' EHR through a hybrid Topic Recurrent Neural Network (TopicRNN) model and GRU. (Dieng et al., 2017).

## 2 Hypotheses chosen for testing

These are the claims/hypotheses we are trying to verify in this project.

- *The proposed model outperforms state-of-the-art methods in readmission prediction (e.g. 0.6103 ± 0.0130 vs. second best 0.5998 ± 0.0124 in terms of ROC-AUC)*

  To test the first hypothesis in its entirety, we will need to build Word2vec+LR, Med2vec+LR, GRU, GRU+Word2Vec, RETAIN, CONTENT as mentioned in the table 2 of (Xiao et al., 2018). For this course project, we are going to work on building only the CONTENT model and evaluate the performance metrics. We are not going to build the other benchmark models that CONTENT model was originally compared against in the paper.

- They claim that the clinical concepts based context vector can be used for patient embedding and clustering patients based on similarity. *The projection matrix $W_v$ can be thought of as a matrix that embeds the clinical concepts (i.e., medical events) into the low dimensional space. The context vector $\theta$ sampled from the recognition network serves as a distributed representation of the patient's medical history. Then we can represent each patient as the concatenation of the context vector and the final hidden state vector of the RNN. In the empirical studies, we will demonstrate their representation power by clustering patients using these vector representations*

## 3 Planned Ablations

Depending on the feasibility, we planned to do two ablations. Test the CONTENT model by removing either one of the ML components individually (not both at the same time): a) local GRU, b) global context (TopicRNN) aka recognition network. This will help us understand how much each model is contributing to the overall prediction. But after a detailed study of the model, we realized that local GRU is critical for the time series prediction and we performed only the other ablation, where we selectively test the addition of the TopicRNN

component.

As an additional experiment, and time permitting, we planned on trying other complex RNN models we learned this semester. For example, RETAIN, in doing the prediction task. PyHealth module (Zhao et al., 2021) has made RETAIN available through their Python package.

## 4 Reuse of Original Code

The original code provided along with the paper is located at (Xiao). Unfortunately, even though the authors published their code on Github, it is not straightforward to be reused for the following reasons.

- Patient data processing code cannot be reused since the author did not publish the complete original dataset.
- For the Natural language processing/word embedding part of code, the word bag is limited.
- Modeling part of the code cannot be reused directly since this paper was written in 2018 and it used python2 and packages like Theano which are not actively developed as of 2023 and documentation support is quite limited. We rewrote parts of code in python3 using Pyhealth, PyTorch packages as needed.
- Missing code for the so-called Thetha layer (part of Recognition network) and the Expression Layer in the original repository of the CONTENT model. This is part of the central value addition from their paper. Documentation is limited.

## 5 Methods

### 5.1 Model Descriptions

This paper proposes CONTENT model. It is similar to a generative language model. i.e. it tries to learn the underlying sample-generating distributions as opposed to discriminative models like SVM and Random Forest which learn decision boundaries given a set of training samples. It models the set of Patients and their visits as a corpus of documents and events inside each visit as a paragraph in a document specific to a patient. This is a joint model with components described below.

It utilizes two machine learning ideas to achieve its target of predicting readmissions. One is the Recurrent Neural Networks (RNN) with Gated Recurrent Units (GRU), which captures the temporal information for a given patient (also perceived as the local information about the patient i.e. short-term disease progression). The other is a TopicRNN model, which is used to capture the global context (chronic diseases, comorbidities, etc.) of patient conditions by using medical knowledge collected from across all the patient data. They use ML models to learn completely from the electronic health record (EHR) data. The EHR data employed in this paper uses two hierarchical inputs viz. "hospital visits" and "clinical events". ICD-9 codes were used as the medical classification prior to performing embedding task.

Internally, the last hidden state form the GRU and patient specific context vector from the TopicRNN components are fed into a logistic regression layer to predict hospital readmission indicator for each patient. The complete CONTENT model as described in the original paper is reproduced in figure 1.

### 5.2 Data Description

We have 2 data sources prepared after writing the proposal.

- The author published 3,000 synthetic patient data on github, of which 2000 are used in model training, 500 for validation, and 500 for testing. The set of all patient EHRs can be considered as the document corpus and each patient EHR is an individual document.
- We download data from MIMIC3 using pyhealth. Getting the description using InnerMap in pyhealth to create similar data. Please refer to the code in the repo "CONTENT/new/Readmission.ipynb" for details (Li and Kuntumalla).

Intially. we planned on combining data from these 2 data sources into as single data source and doing the training. However, we find the following issues that make it not a good option. As also warned by TAs in feedback for the proposal, merging data in 2 disparate formats is always challenging.

The key issue is that after we use InnerMap to collect descriptions for an event using Pyhealth and ICD9, they seem to not follow the same vocabulary as the 3,000 synthetic patient datasets from the original paper. This is a critical issue since

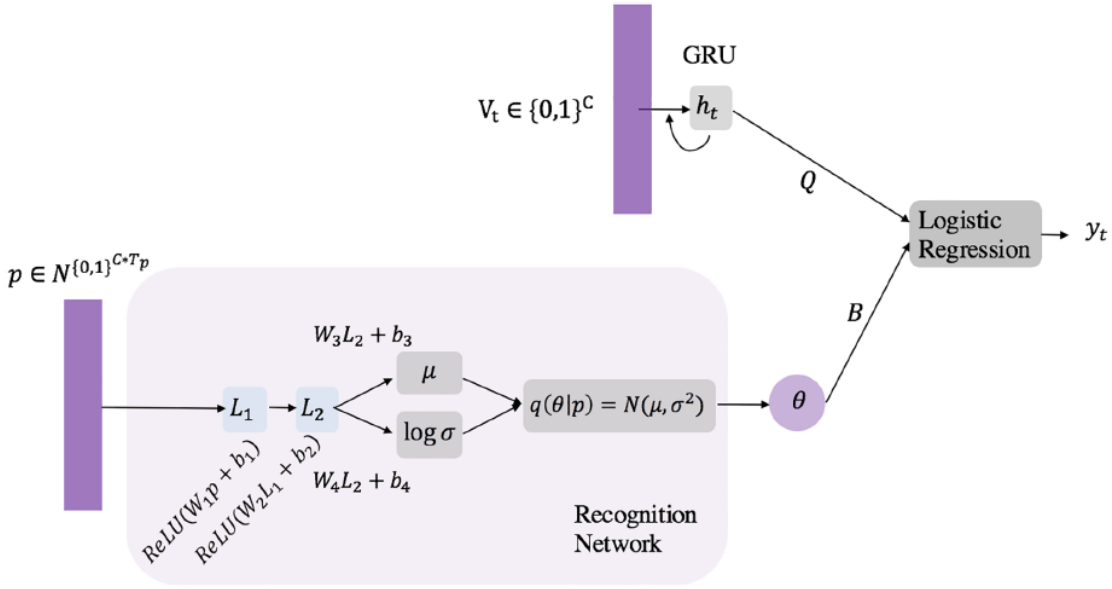


Figure 1: The Content Model (Xiao et al., 2018). This Deep learning model consists of two parts a Timeseries network (GRU) and a Recognition network called Topic RNN.

word bag/mapping/embedding is important in this modeling process. Due to our limited knowledge of health care and time constraints, we dropped the idea of combining these 2 different datasets. Instead, we use only the synthetic dataset.

Instead, we will use the original synthetic 3,000 patient data which is provided by the author. See table1.

### 5.3 Issue in Pyhealth

There is a tiny issue we would like to discuss further. In pyhealth code, in function pyhealth.tasks.readmission_prediction_mimic3_fn, the readmission label is defined in the following way: # get time difference between current visit and next visit

time_diff = (next_visit.encounter_time − visit.encounter_time).days

readmission_label = 1 if time_diff < time_window else 0

This does not seem correct since we are calculating the number of days between 2 encounter times. It should be calculated as encounter_time - previous_discharge_time.

### 5.4 Model Parameters

The original distinct number of the topics (medical terms identifying a given event) is 619. For a given patient, maximum visit size = 300. If a patient has more visits, it was trimmed down to this level. If a patient had fewer visits, then we extend it to this level and make use of the concept of masks (in collate_fn and forward pass) for training and inference purposes in Pytorch. The visit information is a one-hot encoded vector with a pre-selected topics bag of concepts. Some other hyperparameters used vocab size = 619; hidden layers = 200; n_topics = 50; n_batch = 1; learning rate = 0.05; length of visits = 300; 6 epochs for training; threshold = 0.5 for the binary classification on results from the final Logistic regression layer.

We use Binary Cross Entropy as the loss function. Adam (Kingma and Ba, 2017) as the optimizer.

### 5.5 Computational Requirements

In the original paper, the author did not mention equipment requirements. When we implement the code in python3.7 on a Microsoft Surface Laptop with Windows 11 OS and 16GB RAM, it took more than 12 hours for 1 epoch for training data. Later we implemented C/C++ support for Theano by setting up appropriate internal support packages, post which training time for 1 epoch was reduced to 15 minutes. We do not need to explore additional GPU solutions for speedup since this is fast enough for our use cases.

For the PyTorch implementation, we ran the Jupyter notebook on a Lenovo Thinkpad laptop with 16GB RAM, Processor 11th Gen Intel Core

| Dataset | Synthetic EHR Data |
|---|---|
| # patients | 3, 000 |
| # visits | 239, 936 |
| # events | 685, 482 |
| Avg. # of visits per patient | 79.98 |
| Avg. # of events per patient | 228.49 |
| # of unique event codes | 618 |

Table 1: Basic statistics of Synthetic dataset

i7-1165G7, 2.80GHz, 4 Core(s), 8 Logical Processor(s), Windows 11 OS. Device GPU was not used for training. Training time for our implementation: 10 runs of GRU-only model took roughly 5 hours and 10 runs of the CONTENT model took roughly 6 hours.

## 6 Results

We show the results of our experiments in the following subsections.

### 6.1 Reproduction of Original Code

The author published the original code on GitHub (Xiao). Unfortunately, it is outdated since it was using python2. The whole code is not documented making it difficult to understand. There is no explanation of package dependencies, and some of the packages are not maintained anymore. All of these make it hard to reproduce the code. We list here the major efforts we made when trying to reproduce the original code as follows:

- The author did not state which version of Python and corresponding packages in the code. We assume it is using python2 since it uses a package called "cPickle" which is not used in python3. We tried to run the project in python2.7 but we failed since pip dropped support for python2 in 2021. Now are not able to properly install the package for it on Windows x64 machine with python2.7.

- Then we tried to run the code with various sub-versions of python 3 and reached working setup after experiments. There are 2 main packages that are being used in the code: Theano and Lasagne. Unfortunately, since Theano was not maintained for a long time both packages cannot be properly run with python3.11.

- After a long exploration of all package dependencies(mainly about Theano, Lasagne and Numpy), we were able to find a proper environment for the code:
  - windows 11 x64 platform, python3.7
  - lasagne==0.2.dev1; theano==1.0.5; numpy==1.20.3
  - And some other support packages mentioned in the code.

- Another thing we did was to speedup Theano calls by using C/C++ implementation for calculations. Theano will use numpy as default if C/C++ speed up option is not available and will be much slower. This is achieved by properly installing g++ support. g++ (x86_64-posix-seh-rev2, Built by MinGW-W64 project) 12.2.0. As a result, one epoch of training time is reduced from 20+ CPU hours to 20 minutes. Since it is already fast enough we do not need to approach GPU solution.

Table 2 shows the results we generated after trainign and inferencing the CONTENT model. We can see that roc-auc is higher than the one shown in the original paper. But since we are training and testing using synthetic data only, it is possible that there are some differences. And we did 7 experiments in total with default parameters used by the authors. Please refer to config.py and CONTENT.py for details (Li and Kuntumalla). From the values of mean and std, we can observe that the results are stable across experiments.

Figure 2 shows a sample image of t-distributed stochastic neighbor embedding (t-SNE) generated from a Patient-specific vector representation. We can see colored clusters which flock together. These are 2-dimensional projections from higher dimensional space.

### 6.2 Experiments using Pytorch

In addition to the experiments using original notebook, we also created a Jupyter notebook based on the original code using Pytorch as the machine

| run# | roc_auc | pr_auc | accuracy | precision | recall | f1 |
|---|---|---|---|---|---|---|
| 1 | 0.794 | 0.644 | 0.829 | 0.664 | 0.480 | 0.557 |
| 2 | 0.796 | 0.647 | 0.838 | 0.730 | 0.436 | 0.546 |
| 3 | 0.792 | 0.642 | 0.838 | 0.743 | 0.421 | 0.538 |
| 4 | 0.789 | 0.639 | 0.837 | 0.757 | 0.399 | 0.523 |
| 5 | 0.791 | 0.641 | 0.835 | 0.707 | 0.447 | 0.547 |
| 6 | 0.794 | 0.643 | 0.834 | 0.711 | 0.439 | 0.542 |
| 7 | 0.793 | 0.644 | 0.836 | 0.723 | 0.432 | 0.541 |
| mean | 0.792 | 0.643 | 0.835 | 0.719 | 0.436 | 0.542 |
| std | 0.002 | 0.003 | 0.003 | 0.028 | 0.023 | 0.010 |

Table 2: Evaluation results from multiple runs of the CONTENT model on patient EHR data.

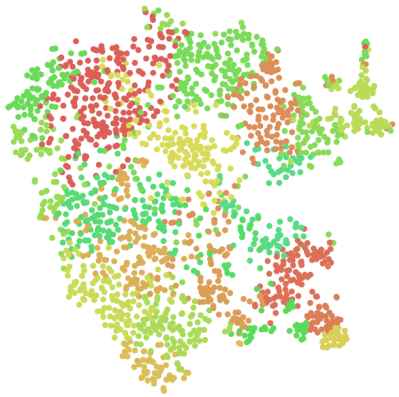

Figure 2: This is an example of clustering generated using T-SNE of Patient-specific vector representation.

learning package. Recalling earlier discussions on the CONTENT model, there are two parts to this model, GRU and TopicRNN. We constructed the code for the GRU component using visit data for each patient. It is similar to a multi-output prediction in time series. The task is to predict if a patient is going to be readmitted into the hospital at each timestamp noted.

Table 3 shows results from using GRU model alone.

Table 4 shows the experimental metrics from experimentation using our implementation of the full CONTENT model which includes the TopicRNN implementation as well.

Comparing the results from PyTorch implementations in tables 3 and 4, we see a marginal improvement in the CONTENT model version. It is not high enough to attribute an exact reason other than that the recognition network added some value to the GRU network.

If the metrics have improved from the GRU model alone, this ablation study would have indicated an improved knowledge addition by using the Recognition network to the GRU model (i.e. global context added to the local context for a given patient). But on the whole, the metrics deteriorated from table 2 (which is the reproduction of original code in original form). This discrepancy likely indicates an incomplete/inefficient replication of the original code in PyTorch. One issue for this was the missing Theta Layer and Expression Layer code in the original model repository where we had to make assumptions while coding (to make the code structure work in Pytorch). Along with missing documentation of these layers from Lasagne to verify the accuracy of translated implementations, these reasons could have led to some issues. Another consideration is that in the PyTorch implementation, we may require different random initialization of weights of various layers, the number of training epochs required, and other hyper parameters discussed in previous sections.

## 7 Conclusion

Overall, we were able to reproduce the code on a synthetic dataset related to the original paper, albeit with some difficulties. When experimenting with data sources, we realized the difficulty in integrating nonstandardized data sources and resorted to using a single data source with the original synthetic dataset provided by the authors. Also along the way, we found a possible bug in the Pyhealth code. We initiated a PyTorch implementation of the original code with mixed results. We were successful in replicating the GRU-only model using Pytorch but we are unable to correctly implement the full CONTENT model using Pytorch. We believe

| run# | roc_auc | pr_auc | accuracy | precision | recall | f1 |
|---|---|---|---|---|---|---|
| 1 | 0.65 | 0.33 | 0.75 | 0.39 | 0.24 | 0.30 |
| 2 | 0.66 | 0.34 | 0.76 | 0.41 | 0.19 | 0.26 |
| 3 | 0.66 | 0.35 | 0.76 | 0.42 | 0.20 | 0.27 |
| 4 | 0.54 | 0.25 | 0.63 | 0.25 | 0.34 | 0.29 |
| 5 | 0.65 | 0.33 | 0.74 | 0.38 | 0.23 | 0.29 |
| 6 | 0.65 | 0.33 | 0.75 | 0.39 | 0.23 | 0.29 |
| 7 | 0.64 | 0.31 | 0.73 | 0.35 | 0.28 | 0.31 |
| 8 | 0.65 | 0.33 | 0.75 | 0.40 | 0.23 | 0.29 |
| 9 | 0.64 | 0.31 | 0.72 | 0.34 | 0.28 | 0.31 |
| 10 | 0.65 | 0.33 | 0.75 | 0.39 | 0.24 | 0.29 |
| mean | 0.64 | 0.32 | 0.73 | 0.37 | 0.25 | 0.29 |
| std | 0.04 | 0.03 | 0.04 | 0.05 | 0.04 | 0.02 |

Table 3: Evaluation results from multiple runs of the GRU part of the CONTENT model on patient EHR data using our code which is a replication using Pytorch instead of Lasagne and Theano.

| run# | roc_auc | pr_auc | accuracy | precision | recall | f1 |
|---|---|---|---|---|---|---|
| 1 | 0.64 | 0.33 | 0.77 | 0.46 | 0.07 | 0.11 |
| 2 | 0.67 | 0.37 | 0.77 | 0.47 | 0.15 | 0.23 |
| 3 | 0.66 | 0.37 | 0.78 | 0.52 | 0.13 | 0.21 |
| 4 | 0.67 | 0.37 | 0.77 | 0.45 | 0.19 | 0.26 |
| 5 | 0.66 | 0.37 | 0.77 | 0.46 | 0.18 | 0.26 |
| 6 | 0.65 | 0.32 | 0.76 | 0.30 | 0.04 | 0.07 |
| 7 | 0.66 | 0.37 | 0.78 | 0.53 | 0.12 | 0.19 |
| 8 | 0.67 | 0.36 | 0.77 | 0.47 | 0.20 | 0.28 |
| 9 | 0.65 | 0.34 | 0.76 | 0.42 | 0.15 | 0.22 |
| 10 | 0.67 | 0.38 | 0.77 | 0.47 | 0.21 | 0.29 |
| mean | 0.66 | 0.36 | 0.77 | 0.46 | 0.14 | 0.21 |
| std | 0.01 | 0.02 | 0.01 | 0.06 | 0.06 | 0.07 |

Table 4: Evaluation results from multiple runs of the CONTENT model (GRU + TopicRNN) on patient EHR data using our code which is a replication using Pytorch instead of Lasagne and Theano.

further efforts in improving the model architecture will yield better results.

## 8 Code Implementation

Please refer to our GitHub repository hosted at (Li and Kuntumalla)

## 9 Communication with the original authors

We sent an email to the original authors requesting clarifications about missing code and original data. We didn't receive any response as of writing this report.

## References


Senjuti Basu Roy, Ankur Teredesai, Kiyana Zolfaghar, Rui Liu, David Hazel, Stacey Newman, and Albert Marinez. 2015. Dynamic hierarchical classification for patient risk-of-readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, page 1691–1700, New York, NY, USA. Association for Computing Machinery.

Adji B. Dieng, Chong Wang, Jianfeng Gao, and John Paisley. 2017. TopicRNN: A recurrent neural network with long-range semantic dependency. In *International Conference on Learning Representations*.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Yiming Li and Gowtham Kuntumalla. Reproduction of content model. `https://github.com/yl159-Yiming/CS598-DL4H`.

Cao Xiao, Tengfei Ma, Adji B. Dieng, David M. Blei, and Fei Wang. 2018. Readmission prediction via deep contextual embedding of clinical concepts. *PLOS ONE*, 13(4):1–15.

Danica Xiao. Content model. https://github.com/danicaxiao/CONTENT.

Yue Zhao, Zhi Qiao, Cao Xiao, Lucas Glass, and Jimeng Sun. 2021. Pyhealth: A python library for health predictive models.