Reproduction of Bidirectional Temporal Encoder Network to Predict Medical Outcomes

GitHub Link:

Team ID: J2
By Nicolas Afonso (nafonso3 | nico.afonso@gatech.edu),
Lucas Lippman (llippman3 | llippman3@gatech.edu)

YouTube Video Presentation: https://youtu.be/gTHHlZJeNck
GitHub Link: https://github.gatech.edu/llippman3/bitenetllippman3nafonso3.git
Presentation Slides: https://docs.google.com/presentation/d/1tWGghPcHKx-RvgT4EH8y27uiUsuDx48LFeejQLcbdIc/edit?usp=sharing

Table of Contents

Introduction

The paper, BiteNet: Bidirectional Temporal Encoder Network to Predict Medical Outcomes, by Xueping Peng, Guodong Long, Tao Shen, Sen Wang, Jing Jiang, and Chengqi Zhang introduces the BiteNet model, a neural network model designed to predict medical outcomes including the probability of readmission into a hospital within 30 days of a previous admission and predicting future diagnoses using electronic health records (EHRs). It uses a modified transformer neural network that uses a bidirectional temporal encoder to address the challenges in processing the hierarchical and temporal nature of EHR data, where patient journeys consist of sequences of medical visits, and each visit containing medical codes (diagnoses and procedures).

While traditional RNN-based models are commonly used for sequential data, they struggle with the longer-term dependencies of medical histories, causing lower performance in predictive tasks. Unlike basic transformer architecture, the BiteNet model introduces an attention mechanism called the *Masked Encoder (MasEnc)*, which captures both contextual and temporal dependencies between patient visits. The model comprises a bidirectional temporal encoder network, using MasEnc to focus on important parts of the patient journey and self-attention mechanisms to process data at three distinct levels: medical codes, visits, and patient journeys.

BiteNet's performance is evaluated against baseline models on both supervised tasks (predicting readmissions and future diagnoses) and unsupervised tasks. The paper shows BiteNet outperforms these models, demonstrating ability in handling long sequences and temporal relationships in EHR records.

Scope of Reproducibility

The original paper focuses its BiteNet model for two scopes: 1) to predict the probability a patient is readmitted to a medical facility within 30 days of admission, and 2) to predict future medical diagnoses. The paper argues that the BiteNet model performs better than the models including: RNN, RETAIN (reverse-time attention RNN), Deepr (CNN), Dipole (RNN), and SAnD.
Model quality is evaluated through several methods including:

1. Supervised prediction of a test data set to get area under the precision recall curve (PR-AUC) for readmission prediction and precision on the predicted top k medical codes that are accurate when predicting future diagnoses.
2. Unsupervised tasks.

Parameterization and model architecture is supported through ablation studies. Several key components of the algorithm were removed or altered to understand their individual impacts on the prediction tasks. Ablation studies were conducted on:

- **Attention pooling:** Evaluated the effect of replacing the two attention pooling layers used to learn representations from medical codes and patient visits with simple summation layers.
- **Direction Masking (DireMask)**: Examined the importance of directional masking in the Masked Encoder component by removing the forward and backward direction masks, determining how the model performed without explicit temporal ordering.

- **Interval Encoding**: Removed interval encodings, which are used to represent the time intervals between patient visits, to assess the effect of temporal distance on the model's ability to make accurate predictions.

The scope of this paper will attempt to reproduce all components of this paper relevant to admissions predictions by improving extraction, transformation, and loading of the dataset, as well as a light exploration of hyperparameter optimization of the BiteNet model. Rather than using more conventional machine learning models as the baseline, our project will utilize and compare PR-AUC performance against Peng et. Al's original BiteNet model. Because we already know the paper's model outperforms baseline models, reproducing this section of the original work serves no constructive purpose. Careful inspection and exploratory data analysis to preprocessing and ETL of the original report is conducted to accelerate and reduce computational load of data processing. We will then re-architect the BiteNet model to allow for simpler configuration of hyperparameters such as batch size, learning rate, weight decay, number of layers, and hidden layer size. This will increase the reproducibility of the report and allow for faster and more exhaustive hyperparameter tuning in future explorations.

Methodology

Dataset Description

There are 3 datasets used practically in this project and 5 datasets used for the construction of our cleaned dataset:

- ADMISSIONS.csv: This dataset includes a subject ID, admission and discharge times with unique event IDs for patients, descriptions of diagnoses reported during the events, and whether the patient died during the event.
- DIAGNOSES_ICD.csv: Links diagnoses codes (in ICD-9 standard) to specific events and the sequences in which they were recorded during the event.
- PROCEDURES_ICD.csv: Contains procedure codes and ties them to an event ID along with the sequence in which they happened during the event.
- PRESCRIPTIONS_ICD.csv
- DRGCODES.csv

PRESCRIPTIONS_ICD.csv and DRGCODES.csv are not used for readmission prediction but are used to construct a cleaned dataset JSON file that mimics the structure of the original report. They are therefore necessary for our project but are not actually used in model training.

Computational Implementation

This project was coded using minimum python version 3.8. Script execution has been conducted entirely in local environments:

1. M2 Chip Mac, 32GB RAM: no issues running the original code or reproduction code
2. 13th Gen Intel I5, 64GB RAM: no issues running the original code or reproduction code

Please visit the README on the root of the GitHub repository for instructions on running the code for this reproduction project.

Data Processing

In attempt to introduce originality to this paper's work, processing and data ETL was conducted independently of the original paper. Pandas and Numpy were used in data processing. Summary statistics were calculated and compared throughout the ETL process to ensure similar preparation. Most data processing is handled by the file *complete.py* and is initialized either by the file *preprocess_mimic.py* or *train_readmission.py*.

'complete.py':

1. Rows with Na values in the columns, drg_code, admission_time, discharge_time, and ndc are removed in the data files DIAGNOSES_ICD.csv, PROCEDURES_ICD.csv, ADMISSIONS.csv, DRGCODES.csv, and PRESCRIPTIONS.csv.
2. Class MimicPreprocessor, filter codes, and filter patients functions set windows for several model parameters. All patients with less than 2 medical visits are removed, all diagnosis codes with less than 5 occurrences are removed.
3. Filter code's function creates bidirectional maps of ICD9 codes to a generated index
4. Create patient journey function:
    a. Goes through each visit sorted by admission time and patient, gets HADMid, event duration (admission duration), and calculates the time between visits as a datetime object, then converting it to a string.
    b. Gets the diagnoses that were conducted during that visit.
    c. Gets the procedures that were conducted during that visit.
    d. Gets the drugs that were prescribed during that visit.
    e. Gets whether the patient died or not.
    f. The function creates a dictionary for each visit with each of these points being a dictionary item. Each dictionary is added to a list called visits.
5. The class PatientJourneyDataset prepares the readmission window of 30 days and provides some training configuration for modeling. The function then calculates the max number of visits across all patients, which is then used for creating fixed-size tensors for batching.
6. Splits the data into an 80/20 train/test split.

'preprocess_mimic.py':

Uses functions from *complete.py* for data processing, filtering and saves the processed data to a JSON file. There are two significant changes to our data processing methodology that differ from the original paper.

The first is that patients are considered readmitted when the days between the last admission date is within 30 days from the previous *discharge* date. In the original project, patients are considered readmitted when the days between the last admission date is within 30 days from the previous *admission* time. Our implementation is more practical as it provides a fixed 30-day window between discharge and the next admission time, whereas the original implementation had a varying window that depended on how long the penultimate medical visit lasted (because the window is calculated from admission time to admission time). Patients with long, second-to-last medical visits had a smaller window between discharge and admission time than patients with a short, second-to-last medical visit.

Second, both processing implementations remove any medical code that occurs less than 5 times in the dataset (known as rare diseases or rare codes). In the original implementation, this is done AFTER patients with less than 2 medical visits are removed, which means some patients with few visits that were diagnosed with a rare disease during a medical visit are included in modeling even though they have less than 2 valid medical visits (visits where some action was taken other than rare code-related action). Our implementation first removes rare codes and then filters patients with less than 2 medical visits. The revisited implementation provides a cleaner and more accurate dataset on which to model on.

‘train_readmission.py’:
Initiates the readmission prediction process (using some functions from complete.py, and functions from BiteNet/model.mh)

Model Description & Code

Our model is implemented using PyTorch as opposed to TensorFlow, the package used in the original study. While we used a different package to architect our model, we relied on the original code in this section. The model architecture is described below in order of execution from start to finish:

- Input: The model takes two main inputs; ‘code inputs’ encodes medical codes including diagnoses and procedures for each patient visit, and ‘interval inputs’ encodes the temporal intervals between visits for each patient.
- Embedding Lookup (*embedding_layer.py*): This embedding layer maps the medical codes and temporal intervals into dense vectors (size 128) to capture the semantic and temporal similarity between them.
- Flattening Layer (*attentionLayers.py*): The multi-dimensional data (medical codes, visit descriptions, and entire journey sequence) is flattened to prepare it for the attention and encoding process. It maintains the patient journey sequence to allow for intra-visit attention.
    - Key differences: The PyTorch version simplifies tensor operations using native methods (meshgrid, unsqueeze) and device management, while the TensorFlow version uses more explicit tf.function decorators and TensorFlow constructs like tf.linalg.diag.
- Intra-Visit Encoding (*common_layer.py*): This file applies an encoder mechanism to process individual visits. The self-attention (4 attention heads) and feed-forward layers capture the relationships among medical codes within a patient visit to learn their relationships and the hierarchical structure of each visit.
    - Key differences: The PyTorch version uses nn.ModuleList for layer management and simplified tensor operations, whereas the TensorFlow version relies on tf.function decorators and TensorFlow-specific layer management through keras.layers.Layer.
- Intra-Visit Attention Pool (*ap_layer.py*): This layer pools the attention across all medical codes in a visit after attention. It focuses only on the most important codes to reduce dimensionality downstream.
    - Key differences: Uses PyTorch-specific tensor operations (unsqueeze, masked_fill) and simpler module structure compared to TensorFlow's custom masking functions and debug statements.
- Position Encoding (*position_encoding_layer.py*): This layer adds positional information to each visit embedding to maintain the temporal sequence of the patient journey. A maximum of 4000 days are considered per patient.

    - Key differences: Our version handles tensor dimensions more explicitly using view() and unsqueeze(), while maintaining the same mathematical logic as the TensorFlow version that uses tf.expand_dims() for dimension management.
- Embedding Layer (*embedding_layer.py*): Shares weights with other embeddings to reduce parameter count.
    - Key differences: Uses PyTorch's built-in nn.Embedding instead of managing custom weights, and removes explicit masking since it's handled by the padding_idx parameter.
- Forward Temporal Encoding (*common_layer.py, Mh_layer.py*): Processes patient journeys in a forward temporal direction (1 layer), where the sequence of visits is processed in the order they occurred.
    - Key differences: Our version uses view() and transpose() for reshaping tensors and handles masking through PyTorch's masked_fill(), while TensorFlow uses split/concat operations and tf.where() for masking, with tensor manipulations being more explicit in the TensorFlow version.
- Forward Attention Pooling (*ap_layer.py):* Pools attention scores across visits in the forward temporal direction to highlight the most influential visits. 0.1 dropout rate is applied.
- Backward Temporal Encoding (1 layer), then Backward Attention Pooling (*common_layer.py and ap_layer.py*).
- FFN (nn.Linear, nn.Dropout, nn.ReLu): Forward and backward representations are concatenated and dimensionality is reduced with a .1 dropout and ReLu function to introduce nonlinearity. Feed-forward network is 512 dimensions.
    - Key differences: The Ffn layer Torch version separates the ReLU activation from the linear layer and uses a dedicated dropout module, while the TensorFlow version combines activation with the dense layer and uses functional dropout.
- Output Layer: Predicts the probability of whether a patient will be readmitted to a medical facility or not.

Main differences are that our model offers more explicit control over components and weight initialization. For example, we use **nn.init.kaiming_normal_** to initialize weights in layers like **nn.Linear and nn.Conv1d,** which allow for finer customization. We prefer PyTorch over Tensorflow as it facilitates debugging and real-time modifications, increasing experimentability.

Parameters

Our best performing model used the following parameters with an Adam optimization function for learning rate optimization and BCEWithLogitsLoss loss function to better handle our imbalanced dataset:

Data:	Model:	Training:
batch_size: 32	embedding_dim: 128	learning_rate: 0.001
num_workers: 2	num_heads: 4	num_epochs: 20
max_visits: 10	num_layers: 1	seed: 112145
max_codes_per_visit: 20	dropout_rate: 0.1	weight_decay: 0.01
	max_days: 4000	max_grad_norm: 1.0
	d_ff: 512	

Evaluation

Our model's performance was compared to the original BiteNet model and evaluated using the precision recall area under the curve score. Our implementation uses an 80/20 train-test split, whereas the original implementation uses an 80/10/10 train-test-validation split. However, we have found that the original implementation did not use the validation set at all, meaning 10% of the dataset was left unused. There was an early-stop function in case there were no improvements to the model over a certain number of epochs, but this function was never passed to the modeling. As such, our model PR-AUC is a more accurate and robust evaluation of performance compared to the original project.

Results

Our project implementation resulted in faster and more efficient compute as well as a higher PR-AUC score. Much of the improvement to our PR-AUC score is due to more robust ETL that has more accurate definitions for 'readmission window' and 'valid medical event'. As such, we include EDA and summaries of our cleaned dataset compared to the original implementation.

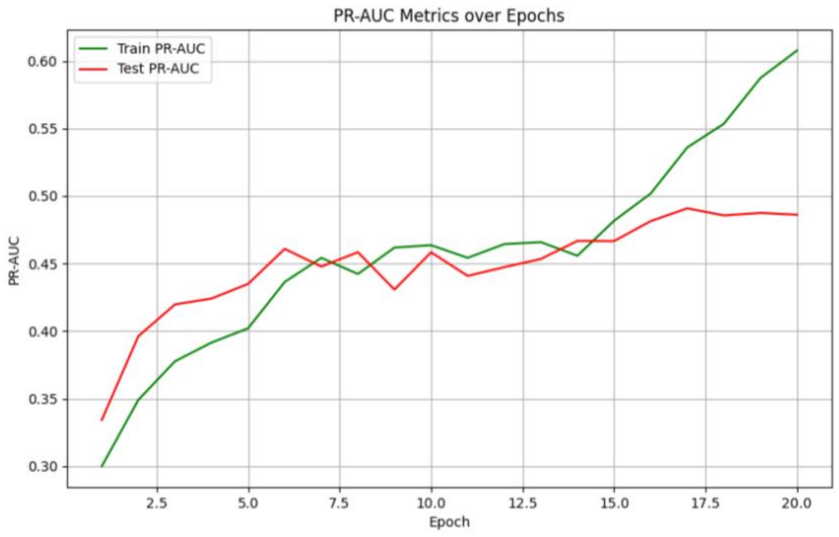
Metric	Original	Revised	Difference	% Change
Total Patients	7,537.00	7,486.00	-51.00	-0.70%
Total Visits	19,993.00	19,882.00	-111.00	-0.60%
Avg Visits/Patient	2.65	2.66	0.01	0.10%
Median Visits/Patient	2.00	2.00	0.00	0.00%
Avg Stay Length	11.19	11.23	0.04	0.40%
Median Stay Length	8.00	8.00	0.00	0.00%
Death Rate %	7.52	7.21	-0.31	-4.10%

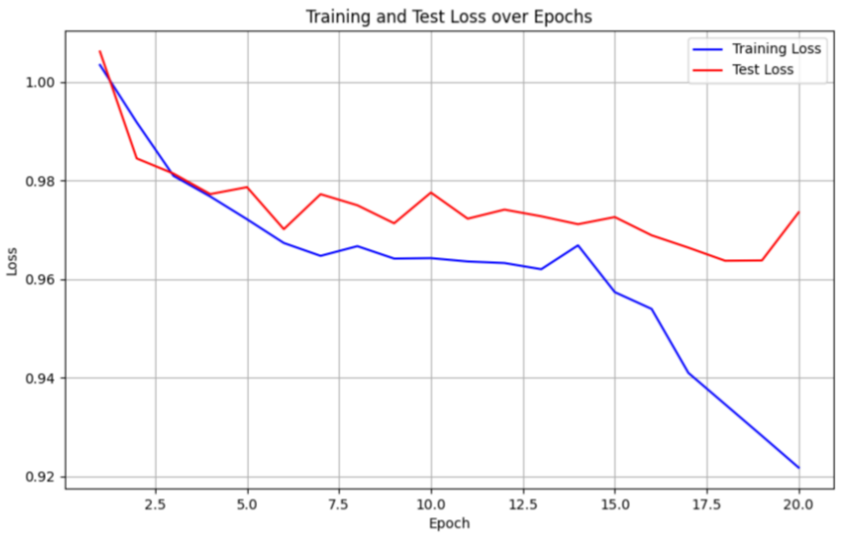
Diagnoses per Patient			
Metric	**Original**	**Revised**	**Difference**
mean	13.02	13.06	0.05
median	11.00	12.00	1.00
std	6.86	6.84	-0.02
min	0.00	1.00	1.00
max	39.00	39.00	0.00
total	260282.00	259747.00	-535.00

CPTs per Patient			
Metric	**Original**	**Revised**	**Difference**
mean	4.09	4.10	0.01
median	3.00	3.00	0.00
std	4.02	4.02	0.00
min	0.00	0.00	0.00
max	40.00	40.00	0.00
total	81755.00	81509.00	-246.00

As can be seen, ETL statistics are very similar among both implementations. However, our implementation removes events where no action was performed, which includes events where patients were diagnosed with a rare condition (appearing less than 5 times in the dataset). We also calculated the 30-day readmission window from discharge to admission rather than from admission to admission. In our training data, readmission cases accounted for 27.77% of our data and 28.24% in our test data. The original paper reports a 20% positive readmission case count in its test dataset. Most of the disparity between dataset distributions is due to our improvements to ETL, as well as some effect from random splitting.

Test PR-AUC in the original paper was reported once as the best score across epochs rather than a PR-AUC for each epoch. After running the model, we realized a best PR-AUC score of 0.3041. Because readmission cases accounted for ~20% of all cases, the original model performed a statistically significant, 50% improvement over guessing randomly (which would be a PR-AUC of 0.20). As can be seen below, our revised paper achieved a best PR-AUC test score of 0.4909 at epoch 17, which is a 61% improvement from the original paper and a 68% improvement from random guessing (since our readmission cases accounted for ~28% of the test dataset).





Discussion

We found our paper, within our defined scope of reproducibility, to be entirely reproducible. The original paper was well-documented and included a complete GitHub repository of the dataset needed, excluding raw data.

While the defined scope of the paper was successfully reproduced and improved-upon, there were several complications that labored reexamination. Firstly, data processing is handled across several files and does not follow the most intuitive methodology; there was little execution or intra-code documentation to follow, which complicated following and understanding the original data pipeline.

Following uninformed reproduction of the ETL, extensive investigation of the original codebase and exploratory analysis of the cleaned datasets were used to identify the differences in data processing methods. This ensured we maintained originality in our data processing, allowing us to introduce mechanisms to accelerate computation and implement a cleaner, more robust process that yielded improved results.

Moreover, through our extensive dissection of the original codebase, we identified numerous logical misunderstandings in the original ETL process which reduced the model's predictive power, such as validating medical events that contained no medical evaluation other than rare diseases (which were meant to be excluded according to the original report's methodology) and defining the readmission window as the time from previous admission to following admission, despite the more common and medically accurate definition of setting the window from previous discharge to the following admission.

While BiteNet's original model implementation was organized and well-separated across separate python files, the usage of TensorFlow complicated readability, debugging, and made implementation more laborious due to TensorFlow having less out-of-box functions and implementations than PyTorch for these types of applications. We believe using PyTorch allows the model to be more easily customizable, especially because we provide a centralized source to make parameter changes in base_config.yaml.

Future reproductions of this research will benefit from our revision not only through an improved and centralized ETL process that is less time-consuming, but also through the model implementation in PyTorch. Future time and research can be dedicated to exploring the model's hyperparameter space more thoroughly; our scope attempted slight changes (no more than 3 different values) to the model's learning rate and batch size, meaning greater effort can be dedicated to maximizing the model's performance.

Finally, because the original BiteNet model was also applied to predicting future drug prescriptions, the improved ETL and revised PyTorch model can be tested for this purpose in the future. While the original BiteNet model outperformed all baseline models, future research could also reintroduce other models with which to compare the updated PyTorch BiteNet, given the changes made to ETL in the scope of this project.

Bibliography

Xueping. “GitHub - Xueping/BiteNet.” *GitHub*, 2020, github.com/Xueping/BiteNet. Accessed 2 Dec. 2024.