Open in Colab

Project Title: HiCu-ICD (Project ID 17) - Final Project Report for CS598 DL4H – Spring 2024

- Project Team ID: **26**
- Team Members: Ratul Saha (**ratuls2 / ratuls2@illinois.edu**)
- Github Repo Link: https://github.com/ratulsaha778/CS598-HiCu-Team26/
- Notebook:
  1. Colab: https://colab.research.google.com/drive/1YC4kPrZb5atKXLMuw4kRxDUKjJSl9mvT?usp=sharing
  2. Github: https://github.com/ratulsaha778/CS598-HiCu-Team26/blob/main/Team26_HiCu_Colab_Notebook_Final.ipynb
- Google Drive link: https://drive.google.com/drive/folders/1wb8_7s_7mVQpuYbco3Em38hf8mxTk7kX?usp=sharing
- Presentation Video:
  - Video 1 (over a prepared deck): https://youtu.be/5l97ffiFtsw
  - Video 2 (over Python Notebook in Google Colab): https://youtu.be/lOPsFnpw-nw

Introduction

The project, titled **"HiCu-ICD", is based on the MLHC 2022 paper "HiCu: Leveraging Hierarchy for Curriculum Learning in Automated ICD Coding"**. HiCu, or Hierarchical Curriculum Learning, improves ICD coding accuracy by leveraging the hierarchy of ICD codes, which groups diagnosis codes based on various organ systems in the human body.

Proceedings of Machine Learning Research 182:1–25, 2022 HiCu: Leveraging hierarchy for curriculum learning in automated ICD coding. W Ren, R Zeng, T Wu, T Zhu, RG Krishnan Machine Learning for Healthcare Conference, 198-223

Google Scholar Link: https://scholar.google.com/citations?view_op=view_citation&hl=en&user=sc_rpHcAAAAJ&citation_for_view=sc_rpHcAAAAJ:u-x6o8ySG0sC

Weiming Ren wren@cs.toronto.edu, Ruijing Zeng jackzeng@cs.toronto.edu, Tongzi Wu tongziwu@cs.toronto.edu, Tianshu Zhu tianshu@cs.toronto.edu, Rahul G. Krishnan rahulgk@cs.toronto.edu (Department of Computer Science, University of Toronto & the Vector Institute, Toronto, Ontario, Canada)

Original Code repository: https://github.com/wren93/HiCu-ICD.

ICD codes are global standardized codes that are used for coding various diagnoses, symptoms, and procedures documented in healthcare settings. They are integral to managing patient care, conducting epidemiological studies, and facilitating healthcare billing. The automation of ICD coding is aimed at improving efficiency and reducing the potential for errors in medical documentation.

Improving clinician throughput is an important technological opportunity for supporting improved healthcare services. When clinicians write notes, a smart process would be beneficial which can document correct diagnosis codes against the human notes. Mapping the long and detailed clinical notes and discharge summaries buried under patient profiles (Electronic Health Records / EHR) to specific ICD (International Classification of Diseases) coding - is a time-consuming, error-prone, and challenging task due to many possible codes and the complex relationships between them. The original paper aims to address the problem of automated coding of medical diagnoses and procedures using the International Classification of Diseases (ICD) system. The paper proposes a novel hierarchical curriculum learning approach that leverages the hierarchical structure of the ICD codes to improve the accuracy and efficiency of automated ICD coding.

The paper uses a hierarchical structure of ICD codes to train the model in a curriculum learning framework. The approach involves learning simpler codes before more complex codes, designed based on the hierarchical structure of the ICD codes. It takes advantage of the hierarchical structure of the ICD codes to improve the model's ability to learn complex relationships between the codes, which is difficult to achieve using traditional flat learning approaches.

Curriculum learning is the design of curricula i.e., in the sequential design of tasks that gradually increase in difficulty. HiCu is an innovation over this process that can predict ICD codes from the natural language descriptions of the patients. It leverages the hierarchy of ICD codes which is grouped based on various organ systems of the human body. The HiCu algorithm uses the graph structure in the space of outputs to design curricula for multi-label classification.

The algorithm is based on:

- The Tree structure: The decision boundaries for different ICD codes are not independent. The ICD codes are organized in a tree structure which defines a notion of similarity between codes. This means that dissimilar labels will have different ancestors in the tree and vice versa. As I go deeper into each sub-tree, the specificity of the codes increases. This means that HiCu can provide wider and non-

overlapping decision boundaries for multi-label classification as the diseases and organs are grouped under defined boundaries.
- HiCu explicitly incorporates techniques to handle label imbalance. This is essential to ensure parity of performance of predictive models on both rare and frequent labels. HiCu can predict rare and frequent labels with equal accuracy.

HiCu is an improvement over curriculum learning which is used in medical code prediction using graph structure for solving multi-label classification problem.

Scope of Reproducibility

ICD codes follow a logical grouping by following a hierarchy of disease, symptoms and body parts. The grouping helps form an ordered tree structure. The HiCu algorithm is inspired from this graph structure which is used to design curricula for multi-label classification. The hierarchical curricula learning claims to provide wider and non-overlapping decision boundary for multi-label classification as the disease and organs are grouped under defined boundaries. ICD codes are organized in a tree structure which establishes similarity, in other words dissimilar labels will have different ancestors in the tree.

As addressed in the original paper, below claims would be validated through reproduction of the model performance results:

- **Claim 1**: HiCuA (Hyperbolic Correction Addition) significantly enhances the MultiResCNN model's performance (improved AUC and F1 scores).
- **Claim 2**: Heirarchical learning model using the graph structure is better than the non-heirarchical representation of ICD codes (degrading AUC and F1 scores).

As part of the project draft, the model was trained on MultiResCNN over 'train_full.csv' data with embed file 'processed_full_100.embed', Asymmetric loss function, and 'HierarchicalHyperbolic' decoder.

The final project report highlights model performance against the original project claims. I ran MultiResCNN with HiCuA. Below programs were used:

- run_multirescnn_50.py
- run_multirescnn_hicua_50.py
- run_multirescnn_hicua_asl_50.py
- run_multirescnn_hicua_asl_full.py
- run_multirescnn_hicuc_50.py
- run_multirescnn_hicuc_asl_50.py

I could not run RAC with HiCuA (below programs) due to resource constraints on Google Colab Pro.

- run_rac_hicua_full.py
- run_rac_50.py
- run_rac_full.py

I kept the LAAT model training out-of-scope for my experiments.

When these programs were run with GPU, I configured the GPU parameter in the code and also updating **parser.add_argument("--gpu"** parameter in **'/utils/options.py'** file. In Google Colab, the programs were run through code snippet:

!python /content/HiCu-ICD-Team26-Spring24/runs/run_multirescnn_hicua_50.py

In local Windows computer, the programs were run through command line:

python /runs/run_multirescnn_hicua_50.py

Methodology

This project aims at improving the ICD classification process. It leverages a hierarchical structure for curriculum learning to improve the accuracy and efficiency of automated ICD coding. It uses the MIMIC-III dataset for model training and evaluation.

The primary objective of this project is to design curricula for multi-label classification models that predict ICD diagnosis and procedure codes from natural language descriptions of patients. The project uses the MIMIC-III dataset for model training and evaluation. The data preprocessing code from MultiResCNN is used to set up the dataset. The project proposes Hierarchical Curriculum Learning (HiCu), an algorithm that uses graph structure in the space of outputs to design curricula for multi-label classification.

Pre-processing was done in local Windows 11 computer. Model training and performance check was done in Google Colab. However I observed, even with Google Colab 1 Tesla T4 GPU, the model training was very slow.

Environment

Team predominantly used Google Colab as the primary platform to train the models. Google Colab with and without GPU was used based on timing and resource availability. Google Colab Pro was used at times to run the model training faster with the help of a GPU. At the same time parallel model trainings sessions were run at local MS Windows 11 computer.

Local windows computer was configured through Anaconda. Original project used **"requirements.txt"** file to setup the environment. It didn't work due to Python and packages' version difference. I used **"environment.yml"** file to configure. Command to configure the environment: conda env create -f environment.yml

Google Colab with GPU configuration:

- python: 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
- gensim: 4.3.2
- nltk: 3.8.1
- numpy: 1.25.2
- pandas: 2.0.3
- scikit-learn: 1.2.2
- scipy: 1.11.4
- tqdm: 4.66.2
- transformers: 4.40.1
- packaging: 24.0
- torch: 2.2.1+cu121
- GPU: Tesla T4. NVIDIA-SMI 535.104.05. Driver Version: 535.104.05
- CUDA Version: 12.2
- High-RAM runtime with 54.8 gigabytes of available RAM
- Number of CPU cores: 8

Data

The dataset MIMIC-III v1.4 (mimic 3) was downloaded from https://physionet.org/content/mimiciii/1.4/ which was first published 2nd Sept 2016 with the intent of enhancing data quality and providing a large amount of additional data for Metavision patients.

Resource citations: Johnson, A., Pollard, T., & Mark, R. (2016). MIMIC-III Clinical Database (version 1.4). PhysioNet. https://doi.org/10.13026/C2XW26.

Original publication: Johnson, A. E. W., Pollard, T. J., Shen, L., Lehman, L. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., & Mark, R. G. (2016). MIMIC-III, a freely accessible critical care database. Scientific Data, 3, 160035.

Citation for PhysioNet: Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. e215–e220.

MIMIC contains detailed information regarding the clinical care of patients. In order to obtain access, it is necessary to finish the MIMIC CITI program training provided by MIT and get the certificate.

- From the website https://www.citiprogram.org/index.cfm?pageID=154&icat=0&ac=0 : Under Register, I needed to use "Massachusetts Institute of Technology Affiliate" as the organization affiliation (not "independent learner") and create an account with UIUC email.
- Then navigated to Massachusetts Institute of Technology Affiliates course. In the Human Subjects training category, I chose "Data or Specimens Only Research" course.\
- I completed the course and got both certificate and report.
- I applied access at PhysioNet: https://eicu-crd.mit.edu/gettingstarted/access/ with UIUC email. Prof. Jimeng Sun and the course (CS 598 Deep Learning for Healthcare at UIUC) was mentioned in the application. Training report was uploaded as certification verification. The actual certificate was not used.

After the application was approved, I downloaded the dataset from the protected website as single large 6.2GB zip file. Over slower internet, it is advisable to download the individual zipped files and unzip them with any unzip / untar tool. I did not have any tool installed in Windows, so I used a python program to unzip them.

This dataset provides a collection of comma-separated value (CSV) files. Each data file has its own identifiers, suffixed with 'ID'. E.g., SUBJECT_ID is assigned to a unique patient, HADM_ID refers to a unique admission, ICUSTAY_ID relates to a unique visit to ICU. Events such as notes, laboratory tests, and fluid balance are stored in a series of 'events' data files. E.g., OUTPUTEVENTS contains all measurements related to output for a given patient, while LABEVENTS contains laboratory test results for a patient. Data files prefixed with 'D_' are dictionaries and contain definitions for identifiers. Rows in CHARTEVENTS linked to a single ITEMID represent the measured concept, but actual name of the measurement is not present in this file. When CHARTEVENTS and D_ITEMS are joined by ITEMID, the details emerge.

The data files used are placed under 'data' folder in project root folder in Google Drive.

data

- D_ICD_DIAGNOSES.csv
- D_ICD_PROCEDURES.csv

- mimic3/PROCEDURES_ICD.csv
- mimic3/DIAGNOSES_ICD.csv
- mimic3/NOTEEVENTS.csv *italicized text*

Supplementary *_hadm_ids.csv files, which contain unique identifiers for hospital admissions, are utilized to ensure that the data for analysis precisely corresponds to specific patient stays. This facilitates accurate matching of clinical notes to their respective admissions, crucial for the integrity of the data used in our experiments.

HADM files were downloaded from https://github.com/jamesmullenbach/caml-mimic/tree/master/mimicdata/mimic3. They are also loaded into Google Drive under mimic3 folder.

- dev_50_hadm_ids.csv
- dev_full_hadm_ids.csv
- test_50_hadm_ids.csv
- test_full_hadm_ids.csv
- train_50_hadm_ids.csv
- train_full_hadm_ids.csv

Together, these resources ensure a comprehensive dataset that supports the experimental replication and validation of the selected hypotheses stated in the original study.

Some statistics about the source data:

- 50K+ patients, including their clinical notes and their corresponding ICD codes.
- 8994 unique ICD-9 codes.
- The experiment used top 50 codes with a subset of 11,317 summaries for training, validation and testing.
- The full dataset has 47,719 training summaries, 1,631 validation summaries and 3,372 testing summaries.
- Notes data has 2,083,180 clinical notes with 92,868,012 tokens.
- 52,726 unique hospital admissions (HADM_ID) and 41,127 unique subjects (SUBJECT_ID).

HiCu algorithm has 2 processes.

- **Pre-processing**: raw files are read, and intermediate files are created.
- **Post-processing**: the model is trained on the intermediate files.

All these data files were preprocessed through a python script 'preprocess_mimic3.py'. The script is executed by running the below command:

```
python preprocess_mimic3.py
```

Pre-processing step needed many Python libraries: genism, nltk, numpy, pandas, scikit_learn, scipy, torch, tqdm, transformers, utils. The inputs csv files were read from the 'data' folder and 'mimic3' sub-folder under 'data', under the project root folder. The output files were saved unde 'mimic3' sub-folder.

The pre-processing **concatenates** clinical notes with their corresponding ICD codes and filtering operations to align medical records correctly This ensures that each clinical note is accurately associated with the correct medical coding. It **removes Rare Terms** during vocabulary building. It lowers the total vocabulary to 51,919 terms from starting 140,796. This step is important for focusing the model's training on relevant terms and avoiding overfitting on noise. The dataset was **split into training, development, and testing sets** for training models.

Below are the high-level steps from the pre-processing program (**preprocess_mimic3.py**):

```
1.  process code-related files
2.  process notes
3.  filter out the codes that not emerge in notes
4.  link notes with their code
5.  statistic unique word, total word, HADM_ID number
6.  split data into train dev test
7.  sort data by its note length, add length to the last column
8.  train word embeddings via word2vec and fasttext
9.  statistic the top 50 code
10. split data according to train_50_hadm_ids dev and test
11. sort data by its note length, add length to the last column
12. create vocab for the models
```

Pre-processing was performed in local Windows computer. Then the processed datasets were loaded into Google Drive. Which then were used in Google Colab during training.

The pre-processing step, for MultiResCNN and RAC models, generates some CSV files (top 50 codes, full set of codes, vocabulary for both models) and processed full 100 and 300 embed files. These files were loaded into Google Drive. While working on the Python Notebook in Google Colab Pro, these files were referenced by mounting Google Drive in Colab.

Model

> Citation of the Original Paper:

Proceedings of Machine Learning Research 182:1–25, 2022 HiCu: Leveraging hierarchy for curriculum learning in automated ICD coding. W Ren, R Zeng, T Wu, T Zhu, RG Krishnan Machine Learning for Healthcare Conference, 198-223

Google Scholar Link: https://scholar.google.com/citations?view_op=view_citation&hl=en&user=sc_rpHcAAAAJ&citation_for_view=sc_rpHcAAAAJ:u-x6o8ySG0sC

Weiming Ren wren@cs.toronto.edu, Ruijing Zeng jackzeng@cs.toronto.edu, Tongzi Wu tongziwu@cs.toronto.edu, Tianshu Zhu tianshu@cs.toronto.edu, Rahul G. Krishnan rahulgk@cs.toronto.edu (Department of Computer Science, University of Toronto & the Vector Institute, Toronto, Ontario, Canada).

> Original Code repository: https://github.com/wren93/HiCu-ICD.

The Curriculum Learning algorithm is described below.

The original paper uses the Poincare model using hyperbolic embeddings. The HiCu architecture consists of an encoder-decoder framework combined with a hierarchical curriculum learning algorithm. The model is first trained on labels at the first level of the label tree using level one decoder, and then proceeds to level two using the knowledge transfer mechanism. This process is repeated until the model reaches the final level in the label tree.

During training at each level, the hyperbolic embeddings of the ICD codes are used to guide the attention computation in the decoder. The hyperbolic embeddings allow the model to learn a representation of the ICD code hierarchy that is more structured and aligned with the hierarchical structure of the codes. The model was run the original HiCu algorithm with high-order grouping of ICD code blocks to create a two-level hierarchy and is trained on the mimic3/train_full.csv dataset to verify its performance.

The model defined in the code includes several classes representing different neural network architectures for automated ICD coding. These models include configurations for handling multi-label classification with a focus on curriculum learning and label hierarchies.

Model Architecture:

- **WordRep** (Word Representation):
  - Embedding Layer: Utilizes pretrained embeddings with a dimension depending on the pretrained file, or initializes a new embedding layer if no pretrained file is provided. The embedding size can be 100, 300, etc., based on the available data.
  - Dropout: Applied after the embedding layer to prevent overfitting, with a dropout rate of 0.1 as configured in the model settings. This helps improve the model's generalization capability on unseen data.
- **Decoders**:
  - RandomlyInitializedDecoder, RACDecoder, LAATDecoder, Decoder: Each uses an attention mechanism tailored to the needs of hierarchical ICD code prediction.
    - Attention Units: Typically involves layers with dimensions tuned to the size of the dataset labels (e.g., number of ICD codes).
    - Activation Function: Uses Tanh or ReLU in intermediate layers to introduce non-linearity.
    - Hyperbolic Embedding Layers: Specific to HiCuA strategies, embedding sizes match the hyperbolic space dimensions used (commonly around 50 dimensions).
- **MultiResCNN**:
  - Convolutional Layers: Multiple convolutional layers with filter sizes that may vary from small (3-5 words) to large (7-9 words) to capture different levels of textual granularity.
  - Residual Connections: Helps in flowing gradients and avoiding the vanishing gradient problem in deep networks.
  - Activation Function: Uses Tanh activation functions following convolutional layers to add non-linearity.
- **LongformerClassifier**:
  - Longformer Layers: Uses a Longformer architecture, suitable for processing long text sequences with attention mechanisms that focus on different parts of the input sequence efficiently.
  - Configuration: Configured with parameters such as number of attention heads, hidden dimensions (typically 768 for base models), and specific attention window sizes.

The model implementation code is present in **/utils/models.py** file.

In [ ]:
```
#Training Code Implementation

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.init import xavier_uniform_ as xavier_uniform
import numpy as np
from utils.utils import build_pretrain_embedding, load_embeddings
from utils.losses import AsymmetricLoss, AsymmetricLossOptimized
from math import floor, sqrt


class WordRep(nn.Module):
```

```
    def __init__(self, args, Y, dicts):
        super(WordRep, self).__init__()

        if args.embed_file:
            print("loading pretrained embeddings from {}".format(args.embed_file))
            if args.use_ext_emb:
                pretrain_word_embedding, pretrain_emb_dim = build_pretrain_embedding(args.embed_file, dicts['w2ind'],
                                                                                     True)
                W = torch.from_numpy(pretrain_word_embedding)
            else:
                W = torch.Tensor(load_embeddings(args.embed_file))

            self.embed = nn.Embedding(W.size()[0], W.size()[1], padding_idx=0)
            self.embed.weight.data = W.clone()
        else:
            # add 2 to include UNK and PAD
            self.embed = nn.Embedding(len(dicts['w2ind']) + 2, args.embed_size, padding_idx=0)
        self.feature_size = self.embed.embedding_dim

        self.embed_drop = nn.Dropout(p=args.dropout)

        self.conv_dict = {1: [self.feature_size, args.num_filter_maps],
                    2: [self.feature_size, 100, args.num_filter_maps],
                    3: [self.feature_size, 150, 100, args.num_filter_maps],
                    4: [self.feature_size, 200, 150, 100, args.num_filter_maps]
                    }


    def forward(self, x):
        features = [self.embed(x)]

        x = torch.cat(features, dim=2)

        x = self.embed_drop(x)
        return x


class RandomlyInitializedDecoder(nn.Module):
    """
    The original per-label attention network: query matrix is randomly initialized
    """
    def __init__(self, args, Y, dicts, input_size):
        super(RandomlyInitializedDecoder, self).__init__()

        Y = Y[-1]

        self.U = nn.Linear(input_size, Y)
        xavier_uniform(self.U.weight)


        self.final = nn.Linear(input_size, Y)
        xavier_uniform(self.final.weight)

        self.loss_function = nn.BCEWithLogitsLoss()


    def forward(self, x, target, text_inputs):
        # attention
        alpha = F.softmax(self.U.weight.matmul(x.transpose(1, 2)), dim=2)

        m = alpha.matmul(x)

        y = self.final.weight.mul(m).sum(dim=2).add(self.final.bias)

        loss = self.loss_function(y, target)
        return y, loss, alpha, m

    def change_depth(self, depth=0):
        # placeholder
        pass


class RACDecoder(nn.Module):
    """
    The decoder proposed by Kim et al. (Code title-guided attention)
    """
    def __init__(self, args, Y, dicts, input_size):
        super(RACDecoder, self).__init__()

        Y = Y[-1]

        self.input_size = input_size

        self.register_buffer("c2title", torch.LongTensor(dicts["c2title"]))
        self.word_rep = WordRep(args, Y, dicts)

        filter_size = int(args.code_title_filter_size)
```

```
        filter_size = int(args.code_title_filter_size)
        self.code_title_conv = nn.Conv1d(self.word_rep.feature_size, input_size,
                                         filter_size, padding=int(floor(filter_size / 2)))
        xavier_uniform(self.code_title_conv.weight)
        self.code_title_maxpool = nn.MaxPool1d(args.num_code_title_tokens)

        self.final = nn.Linear(input_size, Y)
        xavier_uniform(self.final.weight)

        self.loss_function = nn.BCEWithLogitsLoss()

    def forward(self, x, target, text_inputs):
        code_title = self.word_rep(self._buffers['c2title']).transpose(1, 2)
        # attention
        U = self.code_title_conv(code_title)
        U = self.code_title_maxpool(U).squeeze(-1)
        U = torch.tanh(U)

        attention_score = U.matmul(x.transpose(1, 2)) / sqrt(self.input_size)
        alpha = F.softmax(attention_score, dim=2)

        m = alpha.matmul(x)

        y = self.final.weight.mul(m).sum(dim=2).add(self.final.bias)

        loss = self.loss_function(y, target)
        return y, loss, alpha, m

    def change_depth(self, depth=0):
        # placeholder
        pass


class LAATDecoder(nn.Module):
    def __init__(self, args, Y, dicts, input_size):
        super(LAATDecoder, self).__init__()

        Y = Y[-1]

        self.attn_dim = args.attn_dim
        self.W = nn.Linear(input_size, self.attn_dim)
        self.U = nn.Linear(self.attn_dim, Y)
        xavier_uniform(self.W.weight)
        xavier_uniform(self.U.weight)

        self.final = nn.Linear(input_size, Y)
        xavier_uniform(self.final.weight)

        self.loss_function = nn.BCEWithLogitsLoss()

    def forward(self, x, target, text_inputs):
        z = torch.tanh(self.W(x))
        # attention
        alpha = F.softmax(self.U.weight.matmul(z.transpose(1, 2)), dim=2)

        m = alpha.matmul(x)

        y = self.final.weight.mul(m).sum(dim=2).add(self.final.bias)

        loss = self.loss_function(y, target)
        return y, loss, alpha, m

    def change_depth(self, depth=0):
        # placeholder
        pass


class Decoder(nn.Module):
    """
    Decoder: knowledge transfer initialization and hyperbolic embedding correction
    """
    def __init__(self, args, Y, dicts, input_size):
        super(Decoder, self).__init__()

        self.dicts = dicts

        self.decoder_dict = nn.ModuleDict()
        for i in range(len(Y)):
            y = Y[i]
            self.decoder_dict[str(i) + '_' + '0'] = nn.Linear(input_size, y)
            self.decoder_dict[str(i) + '_' + '1'] = nn.Linear(input_size, y)
            xavier_uniform(self.decoder_dict[str(i) + '_' + '0'].weight)
            xavier_uniform(self.decoder_dict[str(i) + '_' + '1'].weight)

        self.use_hyperbolic =  args.decoder.find("Hyperbolic") != -1
        if self.use_hyperbolic:
            self.cat_hyperbolic = args.cat_hyperbolic
```

```
            if not self.cat_hyperbolic:
                self.hyperbolic_fc_dict = nn.ModuleDict()
                for i in range(len(Y)):
                    self.hyperbolic_fc_dict[str(i)] = nn.Linear(args.hyperbolic_dim, input_size)
            else:
                self.query_fc_dict = nn.ModuleDict()
                for i in range(len(Y)):
                    self.query_fc_dict[str(i)] = nn.Linear(input_size + args.hyperbolic_dim, input_size)

            # build hyperbolic embedding matrix
            self.hyperbolic_emb_dict = {}
            for i in range(len(Y)):
                self.hyperbolic_emb_dict[i] = np.zeros((Y[i], args.hyperbolic_dim))
                for idx, code in dicts['ind2c'][i].items():
                    self.hyperbolic_emb_dict[i][idx, :] = np.copy(dicts['poincare_embeddings'].get_vector(code))
                self.register_buffer(name='hb_emb_' + str(i), tensor=torch.tensor(self.hyperbolic_emb_dict[i], dtype=torc

        self.cur_depth = 5 - args.depth
        self.is_init = False
        self.change_depth(self.cur_depth)

        if args.loss == 'BCE':
            self.loss_function = nn.BCEWithLogitsLoss()
        elif args.loss == 'ASL':
            asl_config = [float(c) for c in args.asl_config.split(',')]
            self.loss_function = AsymmetricLoss(gamma_neg=asl_config[0], gamma_pos=asl_config[1],
                                                clip=asl_config[2], reduction=args.asl_reduction)
        elif args.loss == 'ASLO':
            asl_config = [float(c) for c in args.asl_config.split(',')]
            self.loss_function = AsymmetricLossOptimized(gamma_neg=asl_config[0], gamma_pos=asl_config[1],
                                                         clip=asl_config[2], reduction=args.asl_reduction)

    def change_depth(self, depth=0):
        if self.is_init:
            # copy previous attention weights to current attention network based on ICD hierarchy
            ind2c = self.dicts['ind2c']
            c2ind = self.dicts['c2ind']
            hierarchy_dist = self.dicts['hierarchy_dist']
            for i, code in ind2c[depth].items():
                tree = hierarchy_dist[depth][code]
                pre_idx = c2ind[depth - 1][tree[depth - 1]]

                self.decoder_dict[str(depth) + '_' + '0'].weight.data[i, :] = self.decoder_dict[str(depth - 1) + '_' + '0
                self.decoder_dict[str(depth) + '_' + '1'].weight.data[i, :] = self.decoder_dict[str(depth - 1) + '_' + '1

        if not self.is_init:
            self.is_init = True

        self.cur_depth = depth

    def forward(self, x, target, text_inputs):
        # attention
        if self.use_hyperbolic:
            if not self.cat_hyperbolic:
                query = self.decoder_dict[str(self.cur_depth) + '_' + '0'].weight + self.hyperbolic_fc_dict[str(self.cur_
            else:
                query = torch.cat([self.decoder_dict[str(self.cur_depth) + '_' + '0'].weight, self._buffers['hb_emb_' + s
                query = self.query_fc_dict[str(self.cur_depth)](query)
        else:
            query = self.decoder_dict[str(self.cur_depth) + '_' + '0'].weight

        alpha = F.softmax(query.matmul(x.transpose(1, 2)), dim=2)
        m = alpha.matmul(x)

        y = self.decoder_dict[str(self.cur_depth) + '_' + '1'].weight.mul(m).sum(dim=2).add(self.decoder_dict[str(self.cu

        loss = self.loss_function(y, target)

        return y, loss, alpha, m


class ResidualBlock(nn.Module):
    def __init__(self, inchannel, outchannel, kernel_size, stride, use_res, dropout):
        super(ResidualBlock, self).__init__()
        self.left = nn.Sequential(
            nn.Conv1d(inchannel, outchannel, kernel_size=kernel_size, stride=stride, padding=int(floor(kernel_size / 2)),
            nn.BatchNorm1d(outchannel),
            nn.Tanh(),
            nn.Conv1d(outchannel, outchannel, kernel_size=kernel_size, stride=1, padding=int(floor(kernel_size / 2)), bia
            nn.BatchNorm1d(outchannel)
        )

        self.use_res = use_res
        if self.use_res:
            self.shortcut = nn.Sequential(
                        nn.Conv1d(inchannel, outchannel, kernel_size=1, stride=stride, bias=False),
```

```
                    nn.BatchNorm1d(outchannel)
                )

        self.dropout = nn.Dropout(p=dropout)

    def forward(self, x):
        out = self.left(x)
        if self.use_res:
            out += self.shortcut(x)
        out = torch.tanh(out)
        out = self.dropout(out)
        return out


class MultiResCNN(nn.Module):

    def __init__(self, args, Y, dicts):
        super(MultiResCNN, self).__init__()

        self.word_rep = WordRep(args, Y, dicts)

        self.conv = nn.ModuleList()
        filter_sizes = args.filter_size.split(',')

        self.filter_num = len(filter_sizes)
        for filter_size in filter_sizes:
            filter_size = int(filter_size)
            one_channel = nn.ModuleList()
            tmp = nn.Conv1d(self.word_rep.feature_size, self.word_rep.feature_size, kernel_size=filter_size,
                            padding=int(floor(filter_size / 2)))
            xavier_uniform(tmp.weight)
            one_channel.add_module('baseconv', tmp)

            conv_dimension = self.word_rep.conv_dict[args.conv_layer]
            for idx in range(args.conv_layer):
                tmp = ResidualBlock(conv_dimension[idx], conv_dimension[idx + 1], filter_size, 1, True,
                                    args.dropout)
                one_channel.add_module('resconv-{}'.format(idx), tmp)

            self.conv.add_module('channel-{}'.format(filter_size), one_channel)

        if args.decoder == "HierarchicalHyperbolic" or args.decoder == "Hierarchical":
            self.decoder = Decoder(args, Y, dicts, self.filter_num * args.num_filter_maps)
        elif args.decoder == "RandomlyInitialized":
            self.decoder = RandomlyInitializedDecoder(args, Y, dicts, self.filter_num * args.num_filter_maps)
        elif args.decoder == "CodeTitle":
            self.decoder = RACDecoder(args, Y, dicts, self.filter_num * args.num_filter_maps)
        else:
            raise RuntimeError("wrong decoder name")

        self.cur_depth = 5 - args.depth


    def forward(self, x, target, text_inputs):
        x = self.word_rep(x)

        x = x.transpose(1, 2)

        conv_result = []
        for conv in self.conv:
            tmp = x
            for idx, md in enumerate(conv):
                if idx == 0:
                    tmp = torch.tanh(md(tmp))
                else:
                    tmp = md(tmp)
            tmp = tmp.transpose(1, 2)
            conv_result.append(tmp)
        x = torch.cat(conv_result, dim=2)

        y, loss, alpha, m = self.decoder(x, target, text_inputs)

        return y, loss, alpha, m

    def freeze_net(self):
        for p in self.word_rep.embed.parameters():
            p.requires_grad = False

import os
from transformers import LongformerModel, LongformerConfig
class LongformerClassifier(nn.Module):

    def __init__(self, args, Y, dicts):
        super(LongformerClassifier, self).__init__()

        if args.longformer_dir != '':
            print("loading pretrained longformer from {}".format(args.longformer_dir))
```

```
            print("Loading pretrained longformer from {}".format(args.longformer_dir))
            config_file = os.path.join(args.longformer_dir, 'config.json')
            self.config = LongformerConfig.from_json_file(config_file)
            print("Model config {}".format(self.config))
            self.longformer = LongformerModel.from_pretrained(args.longformer_dir, gradient_checkpointing=True)
        else:
            self.config = LongformerConfig(
                attention_mode="longformer",
                attention_probs_dropout_prob=0.1,
                attention_window=[
                    512,
                    512,
                    512,
                    512,
                    512,
                    512,
                ],
                bos_token_id=0,
                eos_token_id=2,
                gradient_checkpointing=False,
                hidden_act="gelu",
                hidden_dropout_prob=0.1,
                hidden_size=768,
                ignore_attention_mask=False,
                initializer_range=0.02,
                intermediate_size=3072,
                layer_norm_eps=1e-05,
                max_position_embeddings=4098,
                model_type="longformer",
                num_attention_heads=12,
                num_hidden_layers=6,
                pad_token_id=1,
                sep_token_id=2,
                type_vocab_size=1,
                vocab_size=50265
            )
            self.longformer = LongformerModel(self.config)

        # decoder
        self.decoder = Decoder(args, Y, dicts, self.config.hidden_size)


    def forward(self, input_ids, token_type_ids, attention_mask, target):
        global_attention_mask = torch.zeros_like(input_ids)
            # global attention on cls token
            # global_attention_mask[:, 0] = 1 # this line should be commented if using decoder
        longformer_output = self.longformer(
            input_ids=input_ids,
            token_type_ids=token_type_ids,
            attention_mask=attention_mask,
            global_attention_mask=global_attention_mask,
            return_dict=False
        )

        output = longformer_output[0]
        y, loss, alpha, m = self.decoder(output, target, None)

        return y, loss, alpha, m

    def freeze_net(self):
        pass


class RACReader(nn.Module):
    def __init__(self, args, Y, dicts):
        super(RACReader, self).__init__()

        self.word_rep = WordRep(args, Y, dicts)
        filter_size = int(args.filter_size)

        self.conv = nn.ModuleList()
        for i in range(args.reader_conv_num):
            conv = nn.Conv1d(self.word_rep.feature_size, self.word_rep.feature_size, kernel_size=filter_size,
                             padding=int(floor(filter_size / 2)))
            xavier_uniform(conv.weight)
            self.conv.add_module(f'conv_{i+1}', conv)

        self.dropout = nn.Dropout(p=args.dropout)

        self.trans = nn.ModuleList()
        for i in range(args.reader_trans_num):
            trans = nn.TransformerEncoderLayer(self.word_rep.feature_size, 1, args.trans_ff_dim, args.dropout, "relu")
            self.trans.add_module(f'trans_{i+1}', trans)

        if args.decoder == "HierarchicalHyperbolic" or args.decoder == "Hierarchical":
            self.decoder = Decoder(args, Y, dicts, self.word_rep.feature_size)
        elif args.decoder == "RandomlyInitialized":
```

```
            self.decoder = RandomlyInitializedDecoder(args, Y, dicts, self.word_rep.feature_size)
        elif args.decoder == "CodeTitle":
            self.decoder = RACDecoder(args, Y, dicts, self.word_rep.feature_size)
        else:
            raise RuntimeError("wrong decoder name")

    def forward(self, x, target, text_inputs=None):
        x = self.word_rep(x)

        x = x.transpose(1, 2)

        for conv in self.conv:
            x = conv(x)

        x = torch.tanh(x).permute(2, 0, 1)
        x = self.dropout(x)

        for trans in self.trans:
            x = trans(x)

        x = x.permute(1, 0, 2)

        y, loss, alpha, m = self.decoder(x, target, text_inputs)

        return y, loss, alpha, m

    def freeze_net(self):
        for p in self.word_rep.embed.parameters():
            p.requires_grad = False


from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
class LAAT(nn.Module):
    def __init__(self, args, Y, dicts):
        super(LAAT, self).__init__()
        self.word_rep = WordRep(args, Y, dicts)

        self.hidden_dim = args.lstm_hidden_dim
        self.biLSTM = nn.LSTM(
            input_size=self.word_rep.feature_size,
            hidden_size=self.hidden_dim,
            batch_first=True,
            dropout=args.dropout,
            bidirectional=True
        )

        self.output_dim = 2 * self.hidden_dim
        self.use_LAAT = False

        self.attn_dim = args.attn_dim
        self.decoder_name = args.decoder
        if "LAAT" in args.decoder:
            if args.decoder == "LAATHierarchicalHyperbolic" or args.decoder == "LAATHierarchical":
                self.decoder_name = args.decoder[4:]
            self.output_dim = self.attn_dim
            self.use_LAAT = True
            self.W = nn.Linear(2 * self.hidden_dim, self.attn_dim)

        if self.decoder_name == "HierarchicalHyperbolic" or self.decoder_name == "Hierarchical":
            self.decoder = Decoder(args, Y, dicts, self.output_dim)
        elif self.decoder_name == "RandomlyInitialized":
            self.decoder = RandomlyInitializedDecoder(args, Y, dicts, self.output_dim)
        elif self.decoder_name == "CodeTitle":
            self.decoder = RACDecoder(args, Y, dicts, self.output_dim)
        elif self.decoder_name == "LAATDecoder":
            self.decoder = RandomlyInitializedDecoder(args, Y, dicts, self.output_dim)
        else:
            raise RuntimeError("wrong decoder name")



        self.cur_depth = 5 - args.depth

    def forward(self, x, target, text_inputs):
        # lengths = (x > 0).sum(dim=1).cpu()
        x = self.word_rep(x)  # [batch, length, input_size]

        # x = pack_padded_sequence(x, lengths, batch_first=True, enforce_sorted=False)
        x1 = self.biLSTM(x)[0]
        # x1 = pad_packed_sequence(x1, batch_first=True)[0]

        if self.use_LAAT:
            x1 = torch.tanh(self.W(x1))

        y, loss, alpha, m = self.decoder(x1, target, text_inputs)
```

```
        return y, loss, alpha, m


def pick_model(args, dicts):
    ind2c = dicts['ind2c']
    Y = [len(ind2c[i]) for i in range(5)] # total number of ICD codes
    if args.model == 'MultiResCNN':
        model = MultiResCNN(args, Y, dicts)
    elif args.model == 'longformer':
        model = LongformerClassifier(args, Y, dicts)
    elif args.model == 'RACReader':
        model = RACReader(args, Y, dicts)
    elif args.model == 'LAAT':
        model = LAAT(args, Y, dicts)
    else:
        raise RuntimeError("wrong model name")

    if args.test_model:
        model.decoder.change_depth(4)
        sd = torch.load(args.test_model)
        model.load_state_dict(sd)
    if args.tune_wordemb == False:
        model.freeze_net()
    if len(args.gpu_list) == 1 and args.gpu_list[0] != -1: # single card training
        model.cuda()
    elif len(args.gpu_list) > 1: # multi-card training
        model = nn.DataParallel(model, device_ids=args.gpu_list)
        model = model.to(f'cuda:{model.device_ids[0]}')
    return model
```

Loading Pre-trained model

Below code can be used to load MultiResCNN with HiCuA architecture, mostly intended for validation and analysis in interactive mode in Google Colab. Further optimization is recommended for better functioning and compatibility.

The pre-trained model files (.pth) are referenced from **"/models"** folder.

Similarly, RAC with HiCuA and LAAT with HiCuA + ASL can also be pre-trained separately and included in the template code for performance testing.

In [ ]:

```
import torch
import torch.nn as nn
import numpy as np

class WordRep(nn.Module):
    def __init__(self, embed_file, vocab_size, embed_dim, dropout):
        super(WordRep, self).__init__()
        if embed_file:
            embeddings = np.load(embed_file)
            W = torch.from_numpy(embeddings).float()
        else:
            W = torch.randn(vocab_size, embed_dim)
        self.embed = nn.Embedding.from_pretrained(W, freeze=False)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.dropout(self.embed(x))

class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, dropout):
        super(ResidualBlock, self).__init__()
        self.left = nn.Sequential(
            nn.Conv1d(in_channels, out_channels, kernel_size=kernel_size, padding=kernel_size//2, bias=False),
            nn.BatchNorm1d(out_channels),
            nn.ReLU(),
            nn.Conv1d(out_channels, out_channels, kernel_size=kernel_size, padding=kernel_size//2, bias=False),
            nn.BatchNorm1d(out_channels)
        )
        self.shortcut = nn.Sequential(
            nn.Conv1d(in_channels, out_channels, kernel_size=1, stride=1, bias=False),
            nn.BatchNorm1d(out_channels)
        )
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        out = self.left(x) + self.shortcut(x)
        return self.dropout(out)

class MultiResCNN(nn.Module):
    def __init__(self, embed_file, vocab_size, embed_size, num_filters, kernel_sizes, dropout):
        super(MultiResCNN, self).__init__()
        self.word_rep = WordRep(embed_file, vocab_size, embed_size, dropout)
```

```
        self.conv = nn.ModuleList()
        for size in kernel_sizes:
            self.conv.append(ResidualBlock(embed_size, num_filters, size, dropout))

    def forward(self, x):
        x = self.word_rep(x).transpose(1, 2)
        for conv in self.conv:
            x = conv(x)
        return x

def load_model(model_path, embed_file, vocab_size, embed_size, num_filters, kernel_sizes, dropout):
    model = MultiResCNN(embed_file, vocab_size, embed_size, num_filters, kernel_sizes, dropout)
    # model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu')))
    model.eval()
    return model

# Specify paths and parameters
model_path = '/content/HiCu-ICD-Team26-Spring24/models/MultiResCNN_HICUA.pth'
embed_file = '/content/HiCu-ICD-Team26-Spring24/data/mimic3/processed_full_100.w2v.wv.vectors.npy'
vocab_size = 51921
embed_size = 100
num_filters = 50
kernel_sizes = [3, 5, 9, 15, 19, 25]
dropout = 0.2

model = load_model(model_path, embed_file, vocab_size, embed_size, num_filters, kernel_sizes, dropout)
print(model)
```

Training

Training Objectives

- Loss Functions:
  - Binary Cross-Entropy Loss: Used for binary classification tasks such as ICD code prediction from clinical texts.
  - Asymmetric Loss: Customized to handle imbalanced datasets, focusing more on the minority classes which are crucial in medical code predictions.
- Optimizer:
  - Adam Optimizer: Widely used for its efficiency in handling sparse gradients and adaptive learning rate capabilities.

The models were first trained locally on a computer with 16GB of RAM and 128MB of GPU memory. Additionally the data was loaded into Google Drive and model training was done on the cloud using this Google Colab notebook "Team26_Colab_Notebook_1.ipynb".

The training program is run through script 'main.py'. Model hyperparameters were maintained in 'options.py'.

- Depth: 5
- Epochs: 2, 3, 5, 10, 500
- Model: MultiResCNN
- Decoder: Hierarchical Hyperbolic
- Batch Size: 8, 16
- Workers: 1, 8, 16
- Drop Out: 0.2
- Loss Function: ASL (Asymmetric Loss)

Computational Requirements:

As per original paper, Bi-LSTM and MultiResCNN Models were trained on a single NVIDIA Tesla V100 GPU. RAC Reader-based Models required more computational power, utilizing 4 NVIDIA Tesla V100 GPUs for training.

I am using a single Tesla T4 GPU for training. The MultiResCNN has been successfully trained.

Training Log for MultiResCNN with HiCuA

Below is portion of the training log that was procued during the training of MutiResCNN with HiCuA on my local Windows computer.

```
C:\Users\ratuls2\\- - - -\cs598>python runs/run_multirescnn_hicua_50.py
Namespace(MODEL_DIR='./models', DATA_DIR='./data', MIMIC_3_DIR='./data/mimic3',
MIMIC_2_DIR='./data/mimic2', data_path='./data/mimic3/train_full.csv', vocab='./data/mimic3/vocab.csv',
Y='full', version='mimic3', MAX_LENGTH=4096, model='MultiResCNN', decoder='HierarchicalHyperbolic',
filter_size='3,5,9,15,19,25', num_filter_maps=50, conv_layer=1,
embed_file='./data/mimic3/processed_full_100.embed', hyperbolic_dim=50, test_model=None,
use_ext_emb=False, cat_hyperbolic=False, loss='BCE', asl_config='0,0,0', asl_reduction='sum',
n_epochs='2,3,5,6,7', depth=5, dropout=0.2, patience=10, batch_size=8, lr=0.0001, weight_decay=0,
criterion='prec_at_8', gpu='0', num_workers=0, tune_wordemb=True, random_seed=1, thres=0.5,
longformer_dir='', reader_conv_num=2, reader_trans_num=4, trans_ff_dim=1024, num_code_title_tokens=36,
code_title_filter_size=9, lstm_hidden_dim=512, attn_dim=512, scheduler=0.9, scheduler_patience=5,
command='python main.py', gpu_list=[0])
loading lookups...
```

```
Depth 0: 34
Depth 1: 270
Depth 2: 1158
Depth 3: 5137
Depth 4: 8921
Training hyperbolic embeddings...
loading pretrained embeddings from ./data/mimic3/processed_full_100.embed
adding unk embedding
MultiResCNN(
  (word_rep): WordRep(
    (embed): Embedding(51921, 100, padding_idx=0)
    (embed_drop): Dropout(p=0.2, inplace=False)
  )
  (conv): ModuleList(
    (channel-3): ModuleList(
      (baseconv): Conv1d(100, 100, kernel_size=(3,), stride=(1,), padding=(1,))
      (resconv-0): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(3,), stride=(1,), padding=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(3,), stride=(1,), padding=(1,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
    (channel-5): ModuleList(
      (baseconv): Conv1d(100, 100, kernel_size=(5,), stride=(1,), padding=(2,))
      (resconv-0): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(5,), stride=(1,), padding=(2,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(5,), stride=(1,), padding=(2,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
    (channel-9): ModuleList(
      (baseconv): Conv1d(100, 100, kernel_size=(9,), stride=(1,), padding=(4,))
      (resconv-0): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(9,), stride=(1,), padding=(4,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(9,), stride=(1,), padding=(4,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
    (channel-15): ModuleList(
      (baseconv): Conv1d(100, 100, kernel_size=(15,), stride=(1,), padding=(7,))
      (resconv-0): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(15,), stride=(1,), padding=(7,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(15,), stride=(1,), padding=(7,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
```

```
      )
    )
    (channel-19): ModuleList(
      (baseconv): Conv1d(100, 100, kernel_size=(19,), stride=(1,), padding=(9,))
      (resconv-0): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(19,), stride=(1,), padding=(9,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(19,), stride=(1,), padding=(9,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
    (channel-25): ModuleList(
      (baseconv): Conv1d(100, 100, kernel_size=(25,), stride=(1,), padding=(12,))
      (resconv-0): ResidualBlock(
        (left): Sequential(
          (0): Conv1d(100, 50, kernel_size=(25,), stride=(1,), padding=(12,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Tanh()
          (3): Conv1d(50, 50, kernel_size=(25,), stride=(1,), padding=(12,), bias=False)
          (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (shortcut): Sequential(
          (0): Conv1d(100, 50, kernel_size=(1,), stride=(1,), bias=False)
          (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
  )
  (decoder): Decoder(
    (decoder_dict): ModuleDict(
      (0_0): Linear(in_features=300, out_features=34, bias=True)
      (0_1): Linear(in_features=300, out_features=34, bias=True)
      (1_0): Linear(in_features=300, out_features=270, bias=True)
      (1_1): Linear(in_features=300, out_features=270, bias=True)
      (2_0): Linear(in_features=300, out_features=1158, bias=True)
      (2_1): Linear(in_features=300, out_features=1158, bias=True)
      (3_0): Linear(in_features=300, out_features=5137, bias=True)
      (3_1): Linear(in_features=300, out_features=5137, bias=True)
      (4_0): Linear(in_features=300, out_features=8921, bias=True)
      (4_1): Linear(in_features=300, out_features=8921, bias=True)
    )
    (hyperbolic_fc_dict): ModuleDict(
      (0): Linear(in_features=50, out_features=300, bias=True)
      (1): Linear(in_features=50, out_features=300, bias=True)
      (2): Linear(in_features=50, out_features=300, bias=True)
      (3): Linear(in_features=50, out_features=300, bias=True)
      (4): Linear(in_features=50, out_features=300, bias=True)
    )
    (loss_function): BCEWithLogitsLoss()
  )
)
train_instances 47719
dev_instances 1631
test_instances 3372
Total epochs at each level: [2, 3, 5, 6, 7]
Training model at depth 0:
EPOCH 0
epoch finish in 596.55s, loss: 0.2857
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.4733, 0.6131, 0.5658, 0.5885, 0.8732
[MICRO] accuracy, precision, recall, f-measure, AUC
0.6456, 0.8056, 0.7648, 0.7846, 0.9477
rec_at_5: 0.5415
prec_at_5: 0.8893
rec_at_8: 0.7294
prec_at_8: 0.7782
rec_at_15: 0.9449
prec_at_15: 0.5670
```

```
evaluation finish in 22.85s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 1
epoch finish in 583.58s, loss: 0.2359
last epoch: testing on dev and test sets
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.5124, 0.6701, 0.5910, 0.6280, 0.9013
[MICRO] accuracy, precision, recall, f-measure, AUC
0.6671, 0.8445, 0.7605, 0.8003, 0.9568
rec_at_5: 0.5534
prec_at_5: 0.9061
rec_at_8: 0.7468
prec_at_8: 0.7962
rec_at_15: 0.9547
prec_at_15: 0.5734

evaluation finish in 20.58s
file for evaluation: ./data/mimic3/test_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.5097, 0.6748, 0.5841, 0.6262, 0.8887
[MICRO] accuracy, precision, recall, f-measure, AUC
0.6760, 0.8493, 0.7681, 0.8067, 0.9560
rec_at_5: 0.5487
prec_at_5: 0.9071
rec_at_8: 0.7395
prec_at_8: 0.7998
rec_at_15: 0.9525
prec_at_15: 0.5789

saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

Training model at depth 1:
EPOCH 0
epoch finish in 602.27s, loss: 0.0849
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.2137, 0.3604, 0.2532, 0.2975, 0.8768
[MICRO] accuracy, precision, recall, f-measure, AUC
0.5062, 0.8023, 0.5783, 0.6722, 0.9694
rec_at_5: 0.3800
prec_at_5: 0.8806
rec_at_8: 0.5319
prec_at_8: 0.7971
rec_at_15: 0.7243
prec_at_15: 0.6121

evaluation finish in 20.69s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 1
epoch finish in 602.62s, loss: 0.0708
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.2532, 0.4027, 0.2975, 0.3422, 0.9070
[MICRO] accuracy, precision, recall, f-measure, AUC
0.5433, 0.8124, 0.6212, 0.7041, 0.9751
rec_at_5: 0.3885
prec_at_5: 0.8961
rec_at_8: 0.5488
prec_at_8: 0.8191
rec_at_15: 0.7512
prec_at_15: 0.6336

evaluation finish in 21.34s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 2
epoch finish in 604.43s, loss: 0.0660
last epoch: testing on dev and test sets
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.2826, 0.4337, 0.3346, 0.3778, 0.9193
[MICRO] accuracy, precision, recall, f-measure, AUC
0.5602, 0.8099, 0.6450, 0.7181, 0.9775
```

```
0.9602, 0.8699, 0.6450, 0.7181, 0.9775
rec_at_5: 0.3944
prec_at_5: 0.9051
rec_at_8: 0.5572
prec_at_8: 0.8296
rec_at_15: 0.7627
prec_at_15: 0.6423

evaluation finish in 21.49s
file for evaluation: ./data/mimic3/test_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.2854, 0.4575, 0.3374, 0.3883, 0.9202
[MICRO] accuracy, precision, recall, f-measure, AUC
0.5594, 0.8069, 0.6459, 0.7175, 0.9768
rec_at_5: 0.3845
prec_at_5: 0.9022
rec_at_8: 0.5458
prec_at_8: 0.8309
rec_at_15: 0.7542
prec_at_15: 0.6505

saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

Training model at depth 2:
EPOCH 0
epoch finish in 663.95s, loss: 0.0263
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0978, 0.1784, 0.1148, 0.1397, 0.9067
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4456, 0.7797, 0.5098, 0.6165, 0.9815
rec_at_5: 0.3273
prec_at_5: 0.8558
rec_at_8: 0.4646
prec_at_8: 0.7831
rec_at_15: 0.6423
prec_at_15: 0.6087

evaluation finish in 23.17s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 1
epoch finish in 664.65s, loss: 0.0229
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.1248, 0.2094, 0.1496, 0.1745, 0.9199
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4817, 0.7682, 0.5637, 0.6502, 0.9842
rec_at_5: 0.3367
prec_at_5: 0.8739
rec_at_8: 0.4775
prec_at_8: 0.8015
rec_at_15: 0.6605
prec_at_15: 0.6262

evaluation finish in 22.70s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 2
epoch finish in 670.18s, loss: 0.0217
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.1399, 0.2289, 0.1696, 0.1948, 0.9266
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4923, 0.7702, 0.5771, 0.6598, 0.9855
rec_at_5: 0.3396
prec_at_5: 0.8813
rec_at_8: 0.4823
prec_at_8: 0.8090
rec_at_15: 0.6715
prec_at_15: 0.6359

evaluation finish in 22.96s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 3
epoch finish in 665.78s, loss: 0.0209
```

```
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.1522, 0.2497, 0.1816, 0.2103, 0.9316
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4976, 0.7822, 0.5777, 0.6645, 0.9864
rec_at_5: 0.3423
prec_at_5: 0.8853
rec_at_8: 0.4870
prec_at_8: 0.8134
rec_at_15: 0.6778
prec_at_15: 0.6408

evaluation finish in 21.52s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 4
epoch finish in 662.12s, loss: 0.0203
last epoch: testing on dev and test sets
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.1603, 0.2607, 0.1933, 0.2220, 0.9344
[MICRO] accuracy, precision, recall, f-measure, AUC
0.5024, 0.7742, 0.5887, 0.6688, 0.9867
rec_at_5: 0.3427
prec_at_5: 0.8860
rec_at_8: 0.4880
prec_at_8: 0.8143
rec_at_15: 0.6807
prec_at_15: 0.6430

evaluation finish in 23.00s
file for evaluation: ./data/mimic3/test_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.1712, 0.2853, 0.2053, 0.2388, 0.9336
[MICRO] accuracy, precision, recall, f-measure, AUC
0.5034, 0.7765, 0.5887, 0.6697, 0.9866
rec_at_5: 0.3342
prec_at_5: 0.8850
rec_at_8: 0.4759
prec_at_8: 0.8174
rec_at_15: 0.6725
prec_at_15: 0.6536

saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

Training model at depth 3:
EPOCH 0
epoch finish in 931.92s, loss: 0.0080
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0439, 0.0782, 0.0534, 0.0635, 0.9266
[MICRO] accuracy, precision, recall, f-measure, AUC
0.3853, 0.7347, 0.4476, 0.5563, 0.9882
rec_at_5: 0.2927
prec_at_5: 0.8200
rec_at_8: 0.4108
prec_at_8: 0.7405
rec_at_15: 0.5749
prec_at_15: 0.5815

evaluation finish in 27.13s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 1
epoch finish in 931.13s, loss: 0.0069
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0547, 0.0907, 0.0658, 0.0763, 0.9341
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4101, 0.7437, 0.4775, 0.5816, 0.9893
rec_at_5: 0.2999
prec_at_5: 0.8372
rec_at_8: 0.4228
prec_at_8: 0.7615
rec_at_15: 0.5928
```

```
prec_at_15: 0.6009

evaluation finish in 26.99s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 2
epoch finish in 929.08s, loss: 0.0066
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0620, 0.1009, 0.0757, 0.0865, 0.9380
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4249, 0.7296, 0.5044, 0.5964, 0.9899
rec_at_5: 0.3034
prec_at_5: 0.8465
rec_at_8: 0.4280
prec_at_8: 0.7702
rec_at_15: 0.5999
prec_at_15: 0.6079

evaluation finish in 27.46s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 3
epoch finish in 929.07s, loss: 0.0064
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0661, 0.1056, 0.0803, 0.0912, 0.9406
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4280, 0.7367, 0.5052, 0.5994, 0.9903
rec_at_5: 0.3038
prec_at_5: 0.8481
rec_at_8: 0.4302
prec_at_8: 0.7725
rec_at_15: 0.6045
prec_at_15: 0.6122

evaluation finish in 26.25s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 4
epoch finish in 927.61s, loss: 0.0062
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0698, 0.1112, 0.0845, 0.0960, 0.9417
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4314, 0.7393, 0.5088, 0.6028, 0.9905
rec_at_5: 0.3053
prec_at_5: 0.8502
rec_at_8: 0.4333
prec_at_8: 0.7774
rec_at_15: 0.6073
prec_at_15: 0.6149

evaluation finish in 26.22s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 5
epoch finish in 927.58s, loss: 0.0061
last epoch: testing on dev and test sets
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0719, 0.1147, 0.0854, 0.0979, 0.9423
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4323, 0.7493, 0.5054, 0.6037, 0.9906
rec_at_5: 0.3073
prec_at_5: 0.8533
rec_at_8: 0.4336
prec_at_8: 0.7784
rec_at_15: 0.6089
prec_at_15: 0.6173

evaluation finish in 27.07s
file for evaluation: ./data/mimic3/test_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0772, 0.1334, 0.0933, 0.1098, 0.9434
```

```
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4289, 0.7456, 0.5024, 0.6003, 0.9905
rec_at_5: 0.2973
prec_at_5: 0.8492
rec_at_8: 0.4195
prec_at_8: 0.7773
rec_at_15: 0.5961
prec_at_15: 0.6237

saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

Training model at depth 4:
EPOCH 0
epoch finish in 1243.63s, loss: 0.0046
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0392, 0.0643, 0.0481, 0.0550, 0.9424
[MICRO] accuracy, precision, recall, f-measure, AUC
0.3676, 0.7115, 0.4319, 0.5375, 0.9899
rec_at_5: 0.2834
prec_at_5: 0.8044
rec_at_8: 0.3958
prec_at_8: 0.7256
rec_at_15: 0.5537
prec_at_15: 0.5691

evaluation finish in 36.42s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 1
epoch finish in 1238.86s, loss: 0.0043
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0457, 0.0726, 0.0559, 0.0632, 0.9449
[MICRO] accuracy, precision, recall, f-measure, AUC
0.3805, 0.7085, 0.4512, 0.5513, 0.9904
rec_at_5: 0.2870
prec_at_5: 0.8132
rec_at_8: 0.4013
prec_at_8: 0.7344
rec_at_15: 0.5625
prec_at_15: 0.5791

evaluation finish in 37.77s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 2
epoch finish in 1240.62s, loss: 0.0041
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0471, 0.0755, 0.0571, 0.0650, 0.9460
[MICRO] accuracy, precision, recall, f-measure, AUC
0.3804, 0.7193, 0.4467, 0.5511, 0.9906
rec_at_5: 0.2871
prec_at_5: 0.8155
rec_at_8: 0.4047
prec_at_8: 0.7406
rec_at_15: 0.5663
prec_at_15: 0.5833

evaluation finish in 37.89s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 3
epoch finish in 1243.17s, loss: 0.0040
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0496, 0.0779, 0.0599, 0.0677, 0.9471
[MICRO] accuracy, precision, recall, f-measure, AUC
0.3866, 0.7153, 0.4570, 0.5577, 0.9907
rec_at_5: 0.2875
prec_at_5: 0.8169
rec_at_8: 0.4045
prec_at_8: 0.7413
rec_at_15: 0.5672
prec_at_15: 0.5852
```

```
evaluation finish in 36.17s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 4
epoch finish in 1242.55s, loss: 0.0039
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0517, 0.0808, 0.0627, 0.0706, 0.9471
[MICRO] accuracy, precision, recall, f-measure, AUC
0.3912, 0.7052, 0.4677, 0.5624, 0.9908
rec_at_5: 0.2890
prec_at_5: 0.8188
rec_at_8: 0.4048
prec_at_8: 0.7421
rec_at_15: 0.5685
prec_at_15: 0.5860

evaluation finish in 38.04s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 5
epoch finish in 1239.93s, loss: 0.0039
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0533, 0.0832, 0.0644, 0.0726, 0.9468
[MICRO] accuracy, precision, recall, f-measure, AUC
0.3917, 0.7112, 0.4658, 0.5629, 0.9906
rec_at_5: 0.2887
prec_at_5: 0.8188
rec_at_8: 0.4061
prec_at_8: 0.7440
rec_at_15: 0.5700
prec_at_15: 0.5870

evaluation finish in 38.00s
saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43

EPOCH 6
epoch finish in 1240.48s, loss: 0.0038
last epoch: testing on dev and test sets
file for evaluation: ./data/mimic3/dev_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0545, 0.0848, 0.0652, 0.0737, 0.9464
[MICRO] accuracy, precision, recall, f-measure, AUC
0.3949, 0.7101, 0.4708, 0.5662, 0.9907
rec_at_5: 0.2894
prec_at_5: 0.8213
rec_at_8: 0.4068
prec_at_8: 0.7458
rec_at_15: 0.5710
prec_at_15: 0.5885

evaluation finish in 35.74s
file for evaluation: ./data/mimic3/test_full.csv

[MACRO] accuracy, precision, recall, f-measure, AUC
0.0626, 0.1049, 0.0768, 0.0887, 0.9470
[MICRO] accuracy, precision, recall, f-measure, AUC
0.3886, 0.7053, 0.4639, 0.5597, 0.9904
rec_at_5: 0.2791
prec_at_5: 0.8187
rec_at_8: 0.3934
prec_at_8: 0.7454
rec_at_15: 0.5546
prec_at_15: 0.5922

saved metrics, params, model to directory ./models\MultiResCNN_HierarchicalHyperbolic_May_02_10_42_43
```

Evaluation

Metrics Used in Evaluation:

- AUC (Area Under the Curve): Utilized in both micro-averaged and macro-averaged forms, this metric measures the overall prediction performance across all labels.
- F1 Score: Reported in both micro-averaged and macro-averaged forms, it indicates the balance between precision and recall.
- Precision@K (P@K): This metric assesses the proportion of correctly predicted labels in the top-K predictions, essential for practical

- Precision@K (P@K): This metric assesses the proportion of correctly predicted labels in the top-K predictions, essential for practical applications where only the top few predictions may be considered.

The new model performance was compared with the original hierarchical model build on CAML, DR-CAML, HyperCore etc.

Performance Results:

- The HiCu method was tested on several model architectures, showing improvements in AUC and F1 scores over baseline models without curriculum learning.
- Notable enhancements were particularly evident for rare labels, addressing the challenge of imbalanced datasets prevalent in medical coding.
- An asymmetric loss function was employed to handle label imbalance more effectively, leading to superior performance on rare and infrequent labels.
- Extensive testing was conducted on the MIMIC-III dataset using ICD-9 codes, establishing the method's effectiveness on a standard dataset for medical coding research.

Below code was used to implement calculation of AUC, F1, and Precision@K metrics.

```
def print_metrics(metrics):
    print()
    if "auc_macro" in metrics.keys():
        print("[MACRO] accuracy, precision, recall, f-measure, AUC")
        print("%.4f, %.4f, %.4f, %.4f, %.4f" % (metrics["acc_macro"], metrics["prec_macro"],
metrics["rec_macro"], metrics["f1_macro"], metrics["auc_macro"]))
    else:
        print("[MACRO] accuracy, precision, recall, f-measure")
        print("%.4f, %.4f, %.4f, %.4f" % (metrics["acc_macro"], metrics["prec_macro"],
metrics["rec_macro"], metrics["f1_macro"]))

    if "auc_micro" in metrics.keys():
        print("[MICRO] accuracy, precision, recall, f-measure, AUC")
        print("%.4f, %.4f, %.4f, %.4f, %.4f" % (metrics["acc_micro"], metrics["prec_micro"],
metrics["rec_micro"], metrics["f1_micro"], metrics["auc_micro"]))
    else:
        print("[MICRO] accuracy, precision, recall, f-measure")
        print("%.4f, %.4f, %.4f, %.4f" % (metrics["acc_micro"], metrics["prec_micro"],
metrics["rec_micro"], metrics["f1_micro"]))
    for metric, val in metrics.items():
        if metric.find("rec_at") != -1:
            print("%s: %.4f" % (metric, val))
    print()

def union_size(yhat, y, axis):
    #axis=0 for label-level union (macro). axis=1 for instance-level
    return np.logical_or(yhat, y).sum(axis=axis).astype(float)

def intersect_size(yhat, y, axis):
    #axis=0 for label-level union (macro). axis=1 for instance-level
    return np.logical_and(yhat, y).sum(axis=axis).astype(float)

def macro_accuracy(yhat, y):
    num = intersect_size(yhat, y, 0) / (union_size(yhat, y, 0) + 1e-10)
    return np.mean(num)

def macro_precision(yhat, y):
    num = intersect_size(yhat, y, 0) / (yhat.sum(axis=0) + 1e-10)
    return np.mean(num)

def macro_recall(yhat, y):
    num = intersect_size(yhat, y, 0) / (y.sum(axis=0) + 1e-10)
    return np.mean(num)

def macro_f1(yhat, y):
    prec = macro_precision(yhat, y)
    rec = macro_recall(yhat, y)
    if prec + rec == 0:
        f1 = 0.
    else:
        f1 = 2*(prec*rec)/(prec+rec)
    return f1


def all_macro(yhat, y):
    return macro_accuracy(yhat, y), macro_precision(yhat, y), macro_recall(yhat, y), macro_f1(yhat, y)

def micro_accuracy(yhatmic, ymic):
    return intersect_size(yhatmic, ymic, 0) / (union_size(yhatmic, ymic, 0) + 1e-10)
```

```
def micro_precision(yhatmic, ymic):
    return intersect_size(yhatmic, ymic, 0) / (yhatmic.sum(axis=0) + 1e-10)

def micro_recall(yhatmic, ymic):
    return intersect_size(yhatmic, ymic, 0) / (ymic.sum(axis=0) + 1e-10)

def micro_f1(yhatmic, ymic):
    prec = micro_precision(yhatmic, ymic)
    rec = micro_recall(yhatmic, ymic)
    if prec + rec == 0:
        f1 = 0.
    else:
        f1 = 2 * (prec * rec) / (prec + rec)
    return f1

def all_micro(yhatmic, ymic):
    return micro_accuracy(yhatmic, ymic), micro_precision(yhatmic, ymic), micro_recall(yhatmic, ymic),
micro_f1(yhatmic, ymic)

from sklearn.metrics import roc_curve, auc
def auc_metrics(yhat_raw, y, ymic):
    if yhat_raw.shape[0] <= 1:
        return
    fpr = {}
    tpr = {}
    roc_auc = {}
    #get AUC for each label individually
    relevant_labels = []
    auc_labels = {}
    for i in range(y.shape[1]):
        #only if there are true positives for this label
        if y[:,i].sum() > 0:
            fpr[i], tpr[i], _ = roc_curve(y[:,i], yhat_raw[:,i])
            if len(fpr[i]) > 1 and len(tpr[i]) > 1:
                auc_score = auc(fpr[i], tpr[i])
                if not np.isnan(auc_score):
                    auc_labels["auc_%d" % i] = auc_score
                    relevant_labels.append(i)

    #macro-AUC: just average the auc scores
    aucs = []
    for i in relevant_labels:
        aucs.append(auc_labels['auc_%d' % i])
    roc_auc['auc_macro'] = np.mean(aucs)

    #micro-AUC: just look at each individual prediction
    yhatmic = yhat_raw.ravel()
    fpr["micro"], tpr["micro"], _ = roc_curve(ymic, yhatmic)
    roc_auc["auc_micro"] = auc(fpr["micro"], tpr["micro"])

    return roc_auc

def recall_at_k(yhat_raw, y, k):
    #num true labels in top k predictions / num true labels
    sortd = np.argsort(yhat_raw)[:,::-1]
    topk = sortd[:,:k]

    #get recall at k for each example
    vals = []
    for i, tk in enumerate(topk):
        num_true_in_top_k = y[i,tk].sum()
        denom = y[i,:].sum()
        vals.append(num_true_in_top_k / float(denom))

    vals = np.array(vals)
    vals[np.isnan(vals)] = 0.

    return np.mean(vals)

def precision_at_k(yhat_raw, y, k):
    #num true labels in top k predictions / k
    sortd = np.argsort(yhat_raw)[:,::-1]
    topk = sortd[:,:k]

    #get precision at k for each example
    vals = []
    for i, tk in enumerate(topk):
        if len(tk) > 0:
            num_true_in_top_k = y[i,tk].sum()
            denom = len(tk)
```

```
                vals.append(num_true_in_top_k / float(denom))

        return np.mean(vals)

    def all_metrics(yhat, y, k=8, yhat_raw=None, calc_auc=True):
        """
            Inputs:
                yhat: binary predictions matrix
                y: binary ground truth matrix
                k: for @k metrics
                yhat_raw: prediction scores matrix (floats)
            Outputs:
                dict holding relevant metrics
        """
        names = ["acc", "prec", "rec", "f1"]

        #macro
        macro = all_macro(yhat, y)

        #micro
        ymic = y.ravel()
        yhatmic = yhat.ravel()
        micro = all_micro(yhatmic, ymic)

        metrics = {names[i] + "_macro": macro[i] for i in range(len(macro))}
        metrics.update({names[i] + "_micro": micro[i] for i in range(len(micro))})

        #AUC and @k
        if yhat_raw is not None and calc_auc:
            #allow k to be passed as int or list
            if type(k) != list:
                k = [k]
            for k_i in k:
                rec_at_k = recall_at_k(yhat_raw, y, k_i)
                metrics['rec_at_%d' % k_i] = rec_at_k
                prec_at_k = precision_at_k(yhat_raw, y, k_i)
                metrics['prec_at_%d' % k_i] = prec_at_k
                metrics['f1_at_%d' % k_i] = 2*(prec_at_k*rec_at_k)/(prec_at_k+rec_at_k)

            roc_auc = auc_metrics(yhat_raw, y, ymic)
            metrics.update(roc_auc)

        return metrics
```

Below Google Colab code snippet was used to:

- Mount Google Drive at Colab.
- Copy complete project folder from Google Drive to Colab for faster processing.
- Model training by executing the respective Python program. Each program has its parameters set inside.

Once the training program execution is complete, the output is saved at **"/models"** folder. It contains **config.csv** (contains the model parameters), **metrics.json** (contains the performance metrics) and **model_best_prec_at_8.pth** (built model).

In [ ]:
```
# Code to mount Google drive:
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# To Copy Google Drive to Google Colab local drive:
!cp -r '/content/drive/MyDrive/HiCu-ICD-Team26-Spring24' '/content/'

# Model training:
# Different training scripts were run with the desired model configuration
!python /content/HiCu-ICD-Team26-Spring24/runs/run_multirescnn_50.py
```

Results

As per the research paper, the HiCu model and its implementation on different encoders showed model performance improvements. AUC, F1 and Precion@K attributes were compared between performance outcomes from various runs on different models. It was observed, when the model was run deep (higher depth) into the hierarchy, and the epoch count increased, it resulted in lesser loss function.

A sample output of the model performance:

```
epoch finish in 189.40s, loss: 0.1643
file for evaluation: ./data/mimic3/dev_50.csv
```

```
[MACRO] accuracy, precision, recall, f-measure, AUC
0.4113, 0.6294, 0.5163, 0.5673, 0.8816
[MICRO] accuracy, precision, recall, f-measure, AUC
0.4590, 0.7112, 0.5641, 0.6292, 0.9161
rec_at_5: 0.5904
prec_at_5: 0.6037
rec_at_8: 0.7229
prec_at_8: 0.4864
rec_at_15: 0.8768
prec_at_15: 0.3299
```

Analyses

Due to large quantity of data and low availability of higher-power GPUs, model was run with a single GPU, and only for few epochs.

I ran the MultiResCNN model with a HierarchicalHyperbolic decoder, that is designed to address the complexities of medical text classificatior My objective was to evaluate the model's effectiveness across various hierarchical depths, with a specific focus on precision at depth 8 (prec_at_8) as the primary criterion. Below is the configuration, and a summary of results across multiple depths. My results are compared with the existing benchmarks from the original paper.

Model Configuration (pertinent to both Claims 1 & 2)

- Model: MultiResCNN with HierarchicalHyperbolic decoder
- Batch size: 8
- Learning rate: 5e-05
- Dropout: 0.2
- Depth: 5
- Random Seed: 1
- Number of Workers: 0
- Filters sizes: filter_size 3,5,9,15,19,25
- Criterion: Precision at depth 8 (prec_at_8)
- Loss: BCE

Models variations available in the Github repo:

- **MultiResCNN** model varieties executed:
  - **Without HiCu**: MultiResCNN_50, MultiResCNN_full
  - **With HiCu**: MultiResCNN_HiCuA_50, MultiResCNN_HiCuA_full,MultiResCNN_HiCuA_asl_50, MultiResCNN_HiCuA_asl_full
- **RAC** model varieties appear to be more resource demanding. I was not been able to run due to resource constraints on Google Colab Pro.
- **LAAT** model varieties were not run due to time constraints.

Result Summary

The MultiResCNN model with HierarchicalHyperbolic decoder, was trained with depth 5. This model used:

- **training data**: './data/mimic3/train_full.csv'
- **vocabulary data**: './data/mimic3/vocab.csv'
- **embed file**: './data/mimic3/processed_full_100.embed'

Below are detailed results for **Claim 1** (attempt to run authors' original algorithm).

Metrics at Depth 0:

- AUC (Macro): 0.8732, 0.9013, 0.8887 (Best: 0.9013)
- AUC (Micro): 0.9477, 0.9568, 0.9560 (Best: 0.9568)
- Loss: 0.2857, 0.2359 (Best: 0.2359)
- F1 (Macro): 0.5885, 0.6280, 0.6262 (Best: 0.6280)
- F1 (Micro): 0.7846, 0.8003, 0.8067 (Best: 0.8067)
- Precision@8: 0.7782, 0.7962, 0.7998 (Best: 0.7998)

Metrics at Depth 1:

- AUC (Macro): 0.8768, 0.9070, 0.9193, 0.9202 (Best: 0.9202)
- AUC (Micro): 0.9694, 0.9751, 0.9775, 0.9768 (Best: 0.9775)
- Loss: 0.0849, 0.0708, 0.0660 (Best: 0.0660)
- F1 (Macro): 0.2975, 0.3422, 0.3778, 0.3883 (Best: 0.3883)
- F1 (Micro): 0.6722, 0.7041, 0.7181, 0.7175 (Best: 0.7181)
- Precision@8: 0.7971, 0.8191, 0.8296, 0.8309 (Best: 0.8309)

Metrics at Depth 2:

- AUC (Macro): 0.9067, 0.9199, 0.9266, 0.9316, 0.9344, 0.9336 (Best: 0.9344)
- AUC (Micro): 0.9815, 0.9842, 0.9855, 0.9864, 0.9867, 0.9866 (Best: 0.9867)
- Loss: 0.0263, 0.0229, 0.0217, 0.0209, 0.0203 (Best: 0.0203)
- F1 (Macro): 0.1397, 0.1745, 0.1948, 0.2103, 0.2220, 0.2388 (Best: 0.2388)
- F1 (Micro): 0.6165, 0.6502, 0.6598, 0.6645, 0.6688, 0.6697 (Best: 0.6697)
- Precision@8: 0.7831, 0.8015, 0.8090, 0.8134, 0.8143, 0.8174 (Best: 0.8174)

Metrics at Depth 3:

- AUC (Macro): 0.9266, 0.9341, 0.9380, 0.9406, 0.9417, 0.9423, 0.9434 (Best: 0.9423)
- AUC (Micro): 0.9882, 0.9893, 0.9899, 0.9903, 0.9905, 0.9906, 0.9905 (Best: 0.9906)
- Loss: 0.0080, 0.0069, 0.0066, 0.0064, 0.0062, 0.0061 (Best: 0.0061)
- F1 (Macro): 0.0635, 0.0763, 0.0865, 0.0912, 0.0960, 0.0979, 0.1098 (Best: 0.1098)
- F1 (Micro): 0.5563, 0.5816, 0.5964, 0.5994, 0.6028, 0.6037, 0.6003 (Best: 0.6037)
- Precision@8: 0.7405, 0.7615, 0.7702, 0.7725, 0.7774, 0.7784, 0.7773 (Best: 0.7784)

Metrics at Depth 4:

- AUC (Macro): 0.9424, 0.9449, 0.9460, 0.9471, 0.9471, 0.9468, 0.9464, 0.9470 (Best: 0.9417)
- AUC (Micro): 0.9899, 0.9904, 0.9906, 0.9907, 0.9908, 0.9906, 0.9907, 0.9904 (Best: 0.9905)
- Loss: 0.0046, 0.0043, 0.0041, 0.0040, 0.0039, 0.0039, 0.0038 (Best: 0.0038)
- F1 (Macro): 0.0550, 0.0632, 0.0650, 0.0677, 0.0706, 0.0726, 0.0737, 0.0887 (Best: 0.0887)
- F1 (Micro): 0.5375, 0.5513, 0.5511, 0.5577, 0.5624. 0.5629, 0.5662, 0.5597 (Best: 0.5662)
- Precision@8: 0.7256, 0.7344, 0.7406, 0.7413, 0.7421, 0.7440, 0.7458, 0.7454 (Best: 0.7458)

Overall Metrics:

Almost all the metrics improve when the depth increases.

- AUC (Macro): 0.9013, 0.9202, 0.9344, **0.9423**, 0.9417 (Best: 0.9423)
- AUC (Micro): 0.9568, 0.9775, 0.9867, **0.9906**, 0.9905 (Best: 0.9906)
- Loss: 0.2359, 0.0660, 0.0203, 0.0061, **0.0038** (Best: 0.0038)
- F1 (Macro): **0.6280**, 0.3883, 0.2388, 0.1098, 0.0887 (Best: 0.6280)
- F1 (Micro): **0.8067**, 0.7181, 0.6697, 0.6037, 0.5662 (Best: 0.8067)
- Precision@8: 0.7998, **0.8309**, 0.8174, 0.7784, 0.7458 (Best: 0.8309)

Claim 1: Metrics Comparison with the Original Paper

I compared my experiment metrics with the numbers mentioned in author's original paper.

- AUC (Macro):
  - Author's MultiResCNN model Metric: 0.9120
  - Author's MultiResCNN with HiCuA Metric: 0.9470
  - Claim 1 MultiResCNN with HiCuA Metric: **0.9423**
- AUC (Micro):
  - Author's MultiResCNN model Metric: 0.9870
  - Author's MultiResCNN with HiCuA Metric: 0.9910
  - Claim 1 MultiResCNN with HiCuA Metric: **0.9906**
- Precision@8:
  - Author's MultiResCNN model Metric: 0.7430
  - Author's MultiResCNN with HiCuA Metric: 0.7480
  - Claim 1 MultiResCNN with HiCuA Metric: **0.8309**

This indeed proves that the hypothesis "Claim 1" stands true: **HiCuA (Hyperbolic Correction Addition) significantly enhances the MultiResCNN model's performance**.

For Claim 2, below are the results (attempt to run modified code with Ablation study with flatten heirarchy).

Claim 2: Metrics Comparison with the Original Paper

I compared my experiment metrics with the numbers mentioned in author's original paper.

- AUC (Macro):
  - Author's MultiResCNN model Metric: 0.9120
  - Author's MultiResCNN with HiCuA Metric: 0.9470
  - Claim 2 MultiResCNN Metric: **0.8983**
  - Claim 2 MultiResCNN with HiCuA Metric: **0.9115**
- AUC (Micro):

  - Author's MultiResCNN model Metric: 0.9870
  - Author's MultiResCNN with HiCuA Metric: 0.9910
  - Claim 2 MultiResCNN Metric: **0.9266**
  - Claim 2 MultiResCNN with HiCuA Metric: **0.9324**
- Precision@8:
  - Author's MultiResCNN model Metric: 0.7430
  - Author's MultiResCNN with HiCuA Metric: 0.7480
  - Claim 2 MultiResCNN Metric: **0.5095**
  - Claim 2 MultiResCNN with HiCuA Metric: **0.5235**

On average, Claim 2 produced results were 6-7% lower performance metrics over Authors' original HiCu algorithm.

This indeed proves that the hypothesis "Claim 2" stands true: **Flatten 2-level heirarchy reduces model formance. Heirarchical learning approach is more efficient than non-heirarchical learning.**

Discussion

Implications of the experimental results, whether the original paper was reproducible, and if it wasn't, what factors made it irreproducible

- Original paper findings were reproducable to the extent I was able to build and run the models through scopes of both Claim 1 and Clain 2.
- However, at this point, I tested only MultiResCNN with and without HICUA, it is premature to conclusively declare the paper's reproducibility for all the referenced models. The initial results align with the paper's claims (as noted in the Results section), suggesting positive reproducibility indications. The ongoing experiments with RAC with HiCuA and LAAT with HiCuA and ASL will provide a comprehensive reproducibility verdict.

What was easy

- Initial setup demanded few hours to determine the compatibility of software package versions (Python and other modules). With "requirements.txt" or "environment.yml" files, it would be easy building the environment for future projects.

What was difficult

- Source data manuverability is touch since MIMIC-III is large.
- Availability of free/low-cost GPU and high-RAM environment was a challenge for me. I wish I had more time to explore other technology options to tackle this limitation. However, some Piazza posts helped garnering those ideas.
- The models' training are resource-intensive and time-consuming. It demanded substantial computational resources which limited the analysis time.
- High cost of powerful GPUs.
- Managing the project alone without referring with atleast another person is tough. Managing the challenges, overcoming the roadblocks and still meeting the project deadline was a daunting task.
- **nltk** and **gensim** might be incompatible on some MAC architectures.
- **NOTEEVENTS.CSV** is a huge file (~4GB). Uploading this file was tough.
- Repeated disconnects at Google Colab made the session recovery impossible, leading to loss of effort and time.

Recommendations to the original authors or others who work in this area for improving reproducibility

- Use a high power (RAM and GPU) local or cloud based computer to traing all the model in parallel. This will help getting all the .pth files. Then one can run performance comparison program components to analyze.
- Understand the software version compatibility upfront to save time.
- Start training with smaller / curated data to build the pipeline (both pre and post processing). Once the smaller set model is confirmed to be built successfully, then introduce large MIMIC-III datasets. This will save some front-load modeling time.
- Use 'environment.yml' file to easily setup modeling environment.
- Utilize cloud based pay-per-use hardware and services.
- Understand pre-requisites to setup inputs to preprocess and train the models. Based on my Python version, I had to make several code conversions from np.int to np.int32 and np.float to np.float32.
- Latest version of packages (**gensim, nltk, numpy, pandas, scikit-learn, scipy, torch, tqdm, transformers**) are recommended and lookout for any functionality deprecation.
- Update command line parameters (**data_path, n_epochs, gpu, workers**) based on CPU/RAM configurations.
- For running all the 500+ iterations mentioned in the original paper, I would suggest to use **Google Cloud Platform** for training the models. **NVIDIA Tesla P100 GPU with CUDA** should be powerful enough to tackle the exhaustive and time-consuming model training process.

Public GitHub Repo

Public GitHub Repo: https://github.com/ratulsaha778/CS598-HiCu-Team26/

README.md: https://github.com/ratulsaha778/CS598-HiCu-Team26/blob/main/README.md

- **CS598-HiCu-Team26**: project folder
  - **README.md**: read me file
  - **main.py**: the master python program to run all the model variations.
  - **preprocess_mimic3.py**: the python program for pre-processing MIMIC-III data
  - **Team26_HiCu_Colab_Notebook_Draft.ipynb**: Python notebook for project draft
  - **Team26_HiCu_Colab_Notebook_Draft.pdf**: PDF of project draft Python notebook
  - **Team26_HiCu_Colab_Notebook_Final.ipynb**: Python notebook for final project (this document)
  - **Team26_HiCu_Colab_Notebook_Draft.pdf**: PDF of project final Python notebook
  - **data**: contains all source files
    - **mimic3**: folder to contain pre-processed MIMIC-III needed to train the model
    - **D_ICD_DIAGNOSES.csv**: diagnosis codes
    - **D_ICD_PROCEDURES.csv**: procedure codes
  - **presentation**: images, video presentation with link, powerpoint presentation.
  - **runs**: python programs each with a different model variation (MultiResCNN, RAC and LAAT).
  - **utils**: utility programs that are referenced throughout the project.
  - **models**: pre-trained models, execution logs, performance metrics.

References

- https://physionet.org/content/mimiciii/1.4/
- https://github.com/wren93/HiCu-ICD/blob/main/README.md
- https://github.com/foxlf823/Multi-Filter-Residual-Convolutional-Neural-Network
- https://paperswithcode.com/paper/hicu-leveraging-hierarchy-for-curriculum
- https://arxiv.org/abs/2208.02301
- https://proceedings.mlr.press/v182/ren22a/ren22a.pdf
- https://github.com/jamesmullenbach/caml-mimic/tree/master/mimicdata/mimic3