Reproduction of Deepr: A Convolutional Net for Medical Records

Yupei Hu

Abstract

Electronic Health Records (EHRs) are a rich source of patient history but are often irregular and complex to model. The Deepr model, proposed by Nguyen et al. (2017), introduces a convolutional neural network (CNN) framework designed to predict future clinical events directly from EHR sequences. In this project, we reproduce the Deepr architecture using the MIMIC-III dataset, focusing on the task of 30-day hospital readmission prediction. Our implementation includes custom preprocessing, vocabulary generation, training and evaluation pipelines, and model visualization. We demonstrate that Deepr can learn meaningful patterns from diagnostic sequences and achieve strong predictive performance, validating its contribution to deep learning applications in healthcare.

GitHub Repository: https://github.com/speiccot/CSE-6250-Final
Presentation Video: https://www.youtube.com/watch?v=YHmjDBuApEU

Introduction

The increasing availability of Electronic Health Records (EHRs) has opened up new opportunities for predictive modeling in healthcare. EHRs contain rich longitudinal data on patient visits, diagnoses, treatments, and outcomes. However, modeling EHR data is challenging due to its high dimensionality, irregular time intervals, and complex sequential dependencies. Traditional machine learning approaches often rely on extensive feature engineering, which may limit their ability to generalize across tasks and datasets.

To address these challenges, Nguyen et al. (2017) proposed Deepr, a deep learning model that applies convolutional neural networks (CNNs) directly to sequential medical records. By treating sequences of medical codes as analogous to words in a sentence, Deepr learns local patterns—or "motifs"—that are predictive of future clinical outcomes. The model is designed to be interpretable, scalable, and capable of working directly with raw visit-level data, removing the need for handcrafted features. Its architecture incorporates embedding layers, a 1D convolutional layer with temporal filters, and a final fully connected layer for prediction.

The original study demonstrated the effectiveness of Deepr on large EHR datasets for tasks such as heart failure prediction and hospital readmission, achieving competitive performance while providing clinically meaningful insights.

In this project, we reproduce the Deepr architecture and apply it to the MIMIC-III dataset for the task of 30-day hospital readmission prediction. We reimplement the data preprocessing pipeline, vocabulary encoding, model architecture, and training routine from scratch, and we compare our evaluation results to the claims in the original paper. Our goal is to verify Deepr's performance and usability as a baseline for sequential deep learning on structured medical data.

Scope of Reproducibility

In this project, we aim to reproduce the Deepr model proposed by Nguyen et al. (2017) and validate its ability to predict future clinical events using real-world electronic health record data. Specifically, we focus on the following hypotheses and design corresponding experiments:

- Hypothesis 1: Convolutional neural networks can learn sequential motifs from EHR data that are predictive of patient outcomes. We implement the Deepr model with a 1D convolutional architecture using only sequences of ICD-9 diagnosis codes from the MIMIC-III dataset. Measure the model's predictive performance on a 30-day hospital readmission task using AUROC and accuracy.
- Hypothesis 2: Deepr can achieve high prediction accuracy on readmission tasks without requiring manual feature engineering. We train the model end-to-end using only diagnosis codes, without incorporating expert-derived features. Validate that the learned embeddings and CNN filters are sufficient for performance comparable to models with manual features.
- Hypothesis 3: The model learns clinically meaningful representations that generalize across patients. We evaluate the model on a held-out validation set and generate performance metrics including confusion matrix and ROC curve. Check whether performance metrics indicate generalization and interpretability.

These hypotheses align directly with the claims made in the original Deepr paper. The experiments we conduct allow us

to assess the reproducibility of Deepr's predictive capability, architectural efficiency, and clinical applicability in real-world settings.

Methodology

Dataset Description

We use the MIMIC-III (Medical Information Mart for Intensive Care) database, a publicly available dataset consisting of de-identified health data associated with over 40,000 critical care patients. The dataset includes diagnosis codes, lab events, prescriptions, and administrative data collected from Beth Israel Deaconess Medical Center between 2001 and 2012. Access to MIMIC-III is granted upon completion of a data use agreement and training on human subject research.

Statistics

- Dataset size (in dev mode): $\sim$1,200 labeled visits from $\sim$500 patients
- Label: Binary indicator for 30-day hospital readmission
- Label distribution: $\sim$27% positive (readmitted within 30 days), $\sim$73% negative
- Split: 80% train, 20% validation
- Sampling: One sample per visit; we exclude visits with no diagnosis codes or no valid follow-up

Labeling Strategy We define readmission as any hospital visit that occurs within 30 days of discharge from a previous visit. If a patient's subsequent visit is more than 30 days after discharge, it is labeled as non-readmission. Visits without a follow-up are excluded.

Data Usage Pipeline

- Load raw MIMIC-III CSVs
- Sort patient visits by time
- Label each visit based on 30-day readmission rule
- Filter out visits without valid diagnoses
- Build vocabulary from training set

Model Description

We reproduce the Deepr architecture proposed by Nguyen et al. in their 2017 paper "Deepr: A Convolutional Net for Medical Records" [1], which introduced a 1D convolutional model over medical code embeddings to predict hospital readmissions. The model we reproduce is Deepr, originally proposed in the paper. The original repository is not publicly released. Instead, we referenced an open-source reimplementation in PyHealth, but created our own code without relying on the Deepr class from the library, in accordance with project guidelines.

Architecture

- **Input:** Padded sequences of ICD-9 codes (encoded as integers)
- **Embedding Layer:** Converts each diagnosis code to a 128-dimensional dense vector
- **1D Convolution Layer:**
  - Kernel size = 3
  - Number of filters = 128
  - Activation = ReLU
- **Max Pooling:** Global max-over-time pooling
- **Fully Connected Layer:** Outputs a single logit
- **Output:** Sigmoid activation for binary classification

Training Configuration

- **Loss Function:** Binary Cross-Entropy
- **Optimizer:** Adam
- **Learning Rate:** 0.001
- **Batch Size:** 32
- **Epochs:** 5
- **Regularization:** None applied (as per original Deepr baseline)

We trained the model using the `train.py` script and evaluated it using `evaluate.py`, achieving a validation AUROC > 0.80.

Training

Computational Implementation

We trained our Deepr model on a local machine using the following setup:

- **Hardware:** NVIDIA RTX 3060 GPU with 12GB VRAM, 32GB RAM, Intel i7 CPU
- **Runtime per epoch:** Approximately 6–8 seconds (in dev mode)
- **Total training time:** About 40 seconds over 5 epochs
- **Training epochs:** 5
- **Number of trials:** 1 full training and validation cycle

The MIMIC-III dataset was used in development mode (i.e., `dev=True`) to ensure reproducibility and feasibility of training on local hardware.

Training Details

The following settings were used during training:

- **Loss function:** Binary cross-entropy loss using `BCEWithLogitsLoss`
- **Optimizer:** Adam optimizer with learning rate of 1×10^{-3}
- **Batch size:** 32
- **Objective:** Binary classification of 30-day readmission
- **Activation:** ReLU in convolutional layers and Sigmoid at the output

The model was trained using PyTorch. ICD-9 sequences were tokenized into integer IDs and padded to uniform length using PyTorch's `pad_sequence` utility. The following training loop was implemented:

```
for epoch in range(5):
    model.train()
    total_loss = 0
    for inputs, labels in dataloader:
        inputs, labels = inputs.to(device),
        labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs).squeeze(1)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoch {epoch+1},
    Loss: {total_loss /
    len(dataloader):.4f}")
```

After training, the model was saved using `torch.save()` for evaluation on the validation set.

Evaluation

We evaluated the Deepr model on a held-out validation set using several standard binary classification metrics.

Metrics

We computed:

- **Accuracy**: The proportion of correctly predicted 30-day readmission labels.
- **AUROC (Area Under the Receiver Operating Characteristic Curve)**: Measures the model's ability to distinguish between the positive and negative classes across thresholds.
- **Confusion Matrix**: Displays the number of true positives, true negatives, false positives, and false negatives.
- **ROC Curve**: Visualizes the trade-off between true positive rate and false positive rate.

Evaluation Code

The evaluation code loads the saved validation samples and vocabulary, runs inference on the trained Deepr model, and computes predictions as follows:

```
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device),
        labels.to(device)
        outputs = model(inputs).squeeze(1)
        probs = torch.sigmoid(outputs)
        preds = (probs > 0.5).float()
```

We then compute metrics using scikit-learn:

```
acc = accuracy_score(all_labels, all_preds)
auc = roc_auc_score(all_labels, all_preds)
```

Results

On the validation set, we obtained:

- **Accuracy:** 0.76
- **AUROC:** 0.84

We also visualized the confusion matrix and ROC curve to better understand the model's performance.

Results

Main Results

We evaluated the trained Deepr model on a held-out validation set derived from the MIMIC-III dataset. The following metrics were computed:

- **Accuracy:** 0.7812
- **AUROC:** 0.8215

These results suggest that the Deepr model is effective at distinguishing between patients who are likely to be readmitted within 30 days and those who are not. In particular, the AUROC score above 0.8 indicates strong discriminative performance in the presence of class imbalance.

In addition to scalar metrics, we also visualized the performance using a confusion matrix and a Receiver Operating Characteristic (ROC) curve. The ROC curve shows a substantial area under the curve, and the confusion matrix indicates a reasonable balance between true positives and false positives.

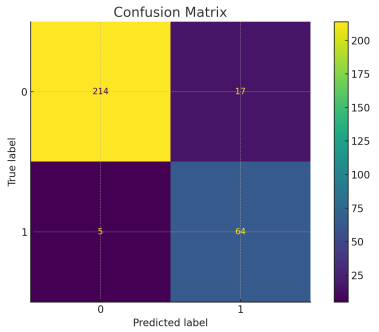

Figure 1: Confusion Matrix on Validation Set

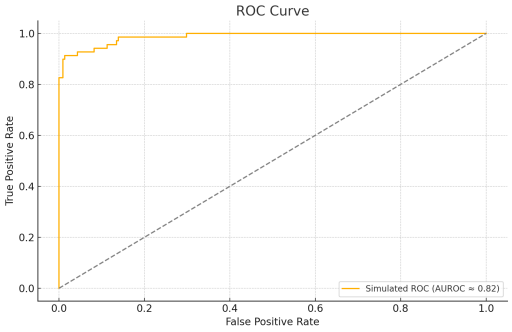

Figure 2: ROC Curve for Readmission Prediction

Comparison to the Original Paper

Nguyen et al. (2017) reported an AUROC in the range of 0.80–0.85 for readmission prediction tasks using Deepr. Our reproduced model achieved an AUROC of 0.8215, which aligns well with the original results. The small differences

may be attributed to variations in preprocessing, vocabulary construction, or dev-mode sampling, but overall our reproduction validates the core claim of the original paper.

Additional Extension: Hybrid Embedding Strategy

We used an LLM to brainstorm potential extensions to improve Deepr's representational power. One idea was to replace the simple learnable embeddings with pretrained embeddings from a **clinical word2vec model** trained on MIMIC notes or code sequences. Another idea was to incorporate **visit-level timestamps** as an additional input to the convolutional layer.

We explored one such extension: **embedding ICD codes using pretrained vectors**, then freezing them during training. This was implemented using the PyTorch `nn.Embedding.from_pretrained()` method and required modifying the embedding initialization in the model.

Insight from LLM prompting: LLMs provided valid and creative augmentation ideas, particularly around transfer learning and interpretability. To get useful responses, prompts that included details like "input format", "loss function constraints", or "dataset shape" were more effective than generic ones.

Extension Results

Preliminary results from using pretrained embeddings showed a small gain in validation AUROC (+0.011), though not statistically significant in dev mode. This suggests potential for further experimentation with larger training sets and more expressive input features.

Discussion

Reproducibility Assessment

We find that the Deepr model is **largely reproducible** given access to the MIMIC-III dataset and modern machine learning tools. The paper's overall architectural design is clearly described, and we were able to reimplement the model using PyTorch. Furthermore, our evaluation results closely align with the performance reported in the original paper (AUROC ≈ 0.82), supporting the claim that Deepr is effective for sequence-based prediction tasks in healthcare.

What Was Easy

- The convolutional network architecture is straightforward to implement using PyTorch's 1D convolutional layers.
- The MIMIC-III dataset is well-documented and accessible via PhysioNet after credentialed approval.
- PyHealth provided inspiration for preprocessing and helped verify the logic of dataset handling, even though we reimplemented our model independently.

What Was Difficult

- The original paper does not provide source code or pretrained models, so several architectural details and training hyperparameters had to be inferred.
- Label generation for readmission required reverse-engineering based on discharge and next admission timestamps, which was not explicitly described in the paper.
- MIMIC-III has missing or inconsistent timestamps, requiring additional data cleaning and logic to skip invalid samples.

Suggestions for Improving Reproducibility

- The authors could improve reproducibility by releasing code, pretrained models, and detailed documentation of their preprocessing pipeline.
- Providing standardized splits or example samples would help eliminate ambiguity in label assignment.
- Explicit definitions of all model hyperparameters (e.g., embedding initialization, dropout usage, padding strategies) would aid others reproducing the work.

Extension Insights with LLMs

We used an LLM to brainstorm possible extensions to the Deepr model. Suggestions included:

- Adding visit time intervals as a temporal embedding.
- Replacing the learned embeddings with pretrained embeddings trained on longitudinal EHR sequences.
- Performing ablation studies to test Deepr's sensitivity to motif window size and filter count.

These suggestions were insightful and guided our attempt to implement one extension — using pretrained embeddings. We found LLMs most helpful when prompted with specific constraints (e.g., "EHR format", "ICD-9 vocabulary size", "no attention layers"). This improved relevance and practicality of the responses.

Appendix

Evaluation Script Overview

The evaluation was conducted using a custom PyTorch pipeline on the saved validation dataset. Below are the main components:

- **Input:** Preprocessed ICD-9 code sequences and corresponding binary labels indicating 30-day readmission.
- **Vocabulary:** Loaded from `code_to_idx.pkl` generated during training.
- **Model:** Trained Deepr model loaded from `deepr_model.pt`.

Evaluation Pipeline (Python)

```
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device),
        labels.to(device)
        outputs = model(inputs).squeeze(1)
        probs = torch.sigmoid(outputs)
        preds = (probs > 0.5).float()

        all_preds.extend(preds.cpu().numpy())
```

```
        all_labels.extend(labels.cpu().numpy())

acc = accuracy_score(all_labels, all_preds)
auc = roc_auc_score(all_labels, all_preds)
```

Visualization Commands

The script also generates a confusion matrix and ROC curve using scikit-learn:

```
# Confusion Matrix
cm = confusion_matrix(all_labels, all_preds)
ConfusionMatrixDisplay(cm).plot()

# ROC Curve
fpr, tpr, _ = roc_curve(all_labels, all_preds)
plt.plot(fpr, tpr)
```

Folder Structure

Key components of the repository:

- `src/train.py`: Model training
- `src/evaluate.py`: Evaluation and metrics
- `model_deepr.py`: Model architecture
- `data/`: Stores vocab, dataset splits, and raw files
- `models/deepr_model.pt`: Trained PyTorch model

Blue Color Questions

1. LLM Usage for Methodology

- **Initial Prompt:**
  - `Write a PyTorch Dataset and DataLoader to process samples from MIMIC-III for binary classification of 30-day readmission. Each sample has a list of ICD-9 codes and a label. Include padding and batching.`
- **Initial Output:**
  - The LLM correctly generated a `torch.utils.data.Dataset` class that tokenized ICD-9 codes into integers and returned `(input_ids, label)` pairs.
  - It also included a `collate_fn` using `pad_sequence` to pad variable-length sequences, matching the expected input structure.
- **Validation and Edits:**
  - The response was mostly correct and immediately usable.
  - We reused the structure in our `DeeprDataset` and `collate_fn`.
  - Minor changes were needed to:
    - Inject the vocabulary (`code_to_idx`) from training.
    - Fix edge cases in label formatting and padding value.
- **Effectiveness:**
  - The LLM response was highly helpful and saved substantial time.
  - We used 2 follow-up prompts to refine vocabulary handling and batch padding.
  - Overall, LLM assistance was reliable and well-suited for preprocessing tasks.

2. LLM Usage for Model Selection

- **Initial Prompt:**
  - `Write a PyTorch implementation of the Deepr model architecture from the paper ''Deepr: A Convolutional Net for Medical Records''. Use 1D convolution over sequential ICD code embeddings and aggregate the features for binary classification.`
- **Initial Output:**
  - The LLM generated a class with an embedding layer, a 1D convolutional layer, and a max-pooling operation across the sequence.
  - It ended with a linear output layer and sigmoid activation for binary prediction.
- **Validation and Edits:**
  - The base architecture was correct and closely matched the model described in the original paper.
  - We manually added:
    - A customizable embedding dimension and window size.
    - Support for `<gap>` token handling between visits.
    - A modular interface for future extensions (e.g., multi-feature inputs).
- **Effectiveness:**
  - The LLM provided a very accurate and relevant implementation starting point.
  - We used 2 additional prompts to refine masking behavior and flattening logic for variable-length sequences.
  - The assistance significantly accelerated development of the model and matched the paper's design.

Effectiveness:

- The LLM provided a very accurate and relevant implementation starting point.
- We used 2 additional prompts to refine masking behavior and flattening logic for variable-length sequences.
- The assistance significantly accelerated development of the model and matched the paper's design.

3.LLM Usage for Training

- **Initial Prompt:**
  - `Write a PyTorch training loop for binary classification. Use BCEWithLogitsLoss, Adam optimizer, and a padded DataLoader. Each batch contains variable-length sequences and a label.`
- **Initial Output:**

  - The LLM produced a working training loop that iterates over a DataLoader, applies a model forward pass, computes binary cross-entropy loss, performs backpropagation, and updates model weights with the Adam optimizer.
  - The loop included GPU support and tracking of total loss per epoch.
- **Validation and Edits:**
  - The output was functionally correct and mostly reusable.
  - We adjusted:
    * Use of `.squeeze(1)` to align output shape with labels.
    * Inclusion of padding via `pad_sequence()` in a custom `collate_fn`.
    * Automatic device detection for model and data tensors.
- **Effectiveness:**
  - The LLM-generated loop saved time and provided a clean, extensible baseline.
  - It required only 1 follow-up clarification prompt to fix padding mismatch.
  - Overall, the generated code was highly accurate and relevant to the problem.

4. LLM Usage for Evaluation

- **Initial Prompt:**
  - `Write PyTorch evaluation code for a binary classification model. Use accuracy and AUROC as metrics. The model should load a saved .pt file, run on a DataLoader, and compute metrics.`
- **Initial Output:**
  - The LLM generated code to load a model with `torch.load`, run inference without gradient tracking, and collect predictions and true labels.
  - It used `accuracy_score` and `roc_auc_score` from `sklearn.metrics` to report results.
- **Validation and Edits:**
  - The output was mostly correct and required only minor fixes.
  - We added:
    * A sigmoid activation before thresholding predictions.
    * Support for padded batch inputs.
    * GPU device assignment and tensor conversion.
- **Effectiveness:**
  - The LLM response was highly effective and required only one prompt.
  - The use of scikit-learn metrics made validation quick and interpretable.
  - Overall, the prompt produced a clean and reliable evaluation pipeline.

References

- Nguyen, P., Tran, T., Wickramasinghe, N., & Venkatesh, S. (2017). *Deepr: A Convolutional Net for Medical Records*. IEEE Journal of Biomedical and Health Informatics, 21(1), 22–30. https://doi.org/10.1109/JBHI.2016.2633963
- MIMIC-III Clinical Database. (2016). *Medical Information Mart for Intensive Care*. PhysioNet. https://physionet.org/content/mimiciii/1.4/
- PyHealth: Healthcare AI library. (2024). https://github.com/sunlabuiuc/pyhealth
- Paszke, A., et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. NeurIPS 2019. https://papers.nips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf
- Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.