Synthetic electronic health records generated with variational graph autoencoders

Giannis Nikolentzos
nikolentzos@lix.polytechnique.fr
École Polytechnique, France

Michalis Vazirgiannis
mvazirg@lix.polytechnique.fr
École Polytechnique, France
and KTH Royal Institute of Technology, Sweden

Christos Xypolopoulos
cxypolop@lix.polytechnique.fr
École Polytechnique, France

Markus Lingman
markus.lingman@regionhalland.se
Institute of Medicine, Sahlgrenska Academy, University of Gothenburg, Sweden

Erik G. Brandt
erik.brandt@shaarpec.com
SHAARPEC, Stockholm, Sweden

Abstract

Data-driven medical care delivery must always respect patient privacy – a requirement that is not easily met. This issue have impeded improvements to healthcare software and have delayed the long-predicted prevalence of artificial intelligence in healthcare. Until now, it has been very difficult to share data between healthcare organizations, resulting in poor statistical models due to unrepresentative patient cohorts. Synthetic data, i. e., artificial but realistic electronic health records, could overcome the drought that is troubling the healthcare sector. Deep neural network architectures in particular have shown an incredible ability to learn from complex data sets, and generate large amounts of unseen data points with the same statistical properties as the training data. Here, we present a generative neural network model that can create synthetic health records with realistic timelines. These clinical trajectories are generated on a per-patient basis and are represented as linear-sequence graphs of clinical events over time. We use a variational graph autoencoder (VGAE) to generate synthetic samples from real-world electronic health records. Our approach generates health records not seen in the training data. We show that these artificial patient trajectories are realistic but still preserve patient privacy, and can therefore be shared freely across organizations.

1 Introduction

Access to real-world health data is often restricted by privacy-protecting regulations like Health Insurance Portability and Accountability Act (HIPAA) and General Data Protection Regulation (GDPR), but also due to technical limitations or simply lacking incentives for data-sharing. Even when pseudo-anonymized (by leaving out personal identifiers such as social security numbers, residence, age, etc), a malicious agent with sufficient knowledge could re-identify patients by connecting patient attributes, conditions, medical prescriptions etc for an individual. Techniques like federated learning [33], differential privacy [1] and homomorphic encryption [2] are actively researched to overcome these barriers.

Carefully created synthetic data could reduce data scarcity by exempting data from privacy preserving regulations. By design, synthetic data mimics real data but is decoupled from real individuals and can be safely shared among healthcare providers, academics, and private stakeholders without leaking sensitive or personally identifiable information. High-quality synthetic data enables exploration and hypothesis generation, but could also be used to pre-train AI models and thus decrease the need for vast amounts of original data. A synthetic data set that mirrors the original data well could also help focusing efforts on more probable hypotheses before seeking confirmation in the source data. Therefore, synthetic data could meet both privacy concerns and re-balance the effort of data access in relation to the chance of relevant findings, and also explore data patterns before investing too much in new research routes.

Healthcare data sets are complex in both space (heterogeneous and strongly connected) and time (cause and effect of symptoms, diagnoses, medications, etc.). Understanding the relationships between the different parts of information about a patient created along a patient trajectory is essential in clinical medicine. It is relatively straightforward to mimic the static properties of a given data distribution, but far more difficult to mimic diverse and coupled time-series with non-equidistant time steps [42, 30]. To the best of our knowledge, this remains to be done in the context of electronic health records (EHRs).

Deep learning (DL) models have revolutionized a wide range of real-world applications, from autonomous vehicles [24], to machine translation [29], and molecule generation in drug discovery [39]. Nevertheless, even the most successful DL model is at the mercy of the amount and quality of its training data set. DL algorithms are very data hungry, and the training samples must adequately reflect the full population that is to be learned. When such conditions can be met, DL algorithms have a canning ability to capture complex data patterns and they also generalize well to unseen data. In practice, available data is often insufficient to train DL models with millions of parameters in any meaningful way. As a consequence, DL models are especially sensitive to limited data availability, as manifested in healthcare.

Machine learning algorithms have already been successfully introduced in the healthcare informatics domain [12, 20, 14, 40, 37, 9, 15]. Variational Autoencoders (VAEs) [22, 23] – and their off-spring graph variational autoencoders [36, 34, 8] – and Generative Adversarial Networks (GANs) [13, 16] are recent deep learning architectures of particular promise. These models learn a "hidden", underlying, data distribution from the training data. VAEs consist of an encoder-decoder pair. The encoder maps the input data to a latent (hidden) distribution, which is randomly sampled by the decoder with the objective to reconstruct the original input data. The latent distribution is usually chosen as a multivariate normal distribution characterized by its mean value and standard deviation. Once the model is trained, an arbitrary number of new samples can be generated by feeding the decoder random samples from the normal distribution. GANs, on the other hand, use two neural networks that are trained together but in adverse. The two networks are known as the generator and the discriminator. The generator learns to create samples as realistic as possible, while the discriminator learns to distinguish synthetic samples from real ones. Once both networks are fully trained, the generator can create unseen data samples with a high similarity to the real data.

Earlier efforts to generate synthetic EHRs have revolved around GANs [12, 4, 41, 3]. Notably, Choi et al. proposed medGAN [12], a neural network model that generates high-dimensional discrete variables to represent EHR events. Baowaly et al. [4] derived two enhanced versions of medGAN with a more complex (Wasserstein) architecture, and Yale et al. [41] identified limitations to medGAN and proposed HealthGAN, another Wasserstein-based method. They also developed improved metrics for synthetic health data quality. Chin-Cheong et al. [10] created synthetic EHR data with GANs trained on patient data from intensive care units. Mimicking a real-world scenario with data sets from different organizations isolated in silos, the final results were combined with federated learning [33]. Finally, Esteban et al. [14] proposed a recurrent GAN to generate synthetic medical time series using recurrent neural networks for both the generator and the discriminator. While GANs have achieved promising results, they tend to be unstable with oscillating model parameters that are hard to train. This problem can be particularly severe for time series, where long-range interactions and order between elements are crucial to learn. Other approaches are Bayesian network learning [21], and deterministic differential modeling e. g., as implemented in the popular open-source software Synthea [38]. This open-source software package is designed to simulate the lifespans of synthetic patients but is based on fixed demographic properties extracted from public data, and does not

learn through a training procedure. On the other hand, Variational Graph Autoencoders (VGAEs) are easy to train, have been applied successfully to several learning problems on graphs, and can accurately model the underlying data distribution.

In this paper, we develop a machine learning algorithm for generating electronic healthcare records represented as sequential graphs (patient trajectories). A patient trajectory is a time sequence of encounters (visits) at healthcare organizations (e. g., hospitals or other providers). Each encounter links to patient interventions such as identified diagnosis and dispensed medications. Analyzing such patient trajectories are key to deliver data-driven insights to healthcare organizations. Creating synthetic EHRs with graph deep learning is to the best of our knowledge a new concept. Synthetic graphs is already a hot topic in drug design [43, 18, 25, 6], but patient trajectories require much larger (e. g., hundreds of nodes) graph representations than their drug molecule counterparts. This poses a significant challenge to generation algorithms. Here, we propose a VGAE tailored to patient trajectories that can generate novel large-scale samples.

2 Results

2.1 EHR data source

Details. The Medical Information Mart for Intensive Care (MIMIC-IV) database [19] was the source to all our experiments. MIMIC-IV provides critical care data for thousands of patients admitted to the intensive care units at the Beth Israel Deaconess Medical Center. We extracted a subset of patients whose trajectories contain any of the ICD-10-CM diagnosis codes I48.0, I48.1, I48.2, and I48.9. This group corresponds to a real-world cohort of patients diagnosed with atrial fibrillation.

A trajectory graph was calculated for each of those 6535 patients. Our data model is a labeled property graph (LPG) that follows the FHIR standard for healthcare data [5]. The model encodes many kinds of healthcare data in graph form, including clinical, capacity, resource, and financial data. In this work, we use a subset of the full model to focus on diagnosis and medications for patients during visits to the emergency unit and the following in-patient stays. This condensed model is a directed graph with labeled nodes and edges where metadata can be stored with key-value pairs on all entities.

Patient trajectories are directed acyclic graphs (DAGs), i. e., they do not contain directed cycles. The node and edge labels are described by the functions ℓ_V and ℓ_E which assign elements from the sets Σ_V and Σ_E (i. e., $\ell_V : V \to \Sigma_V$ and $\ell_E : E \to \Sigma_E$). Here, V and E denote the set of nodes and edges of all graphs, respectively. There are in total $|\Sigma_V| = 13,980$ different node labels and $|\Sigma_E| = 6$ different edge labels. Figure 1 shows an example of a patient trajectory. Each trajectory contains one Patient node, and a number of Encounter nodes that form the patient timeline. Each Encounter is described by an EncounterCategory and is shaped like a star graph with Condition/ConditionType and MedicationRequests/MedicationType pairs for the diagnosis and medication events. The edge labels are ATTENDS between Patient and each Encounter, NEXT between neighboring Encounters, OF_CATEGORY to describe the Encounter, DIAGNOS between Encounter and Condition, ADMINISTRATED between Encounter and Medication, and OF_TYPE to describe the Condition/ConditionType and MedicationRequests/MedicationType pairs. The edge labels are uniquely determined by the label pair of the ancestor and successor nodes. Note that edge labels are omitted from Figure 1 to simplify the presentation. Figure 2 shows the frequency of the node and edge labels in the training data (see Table 1 for more details on the source data). It should be mentioned that the ConditionType and MedicationType labels are higher level labels that contain all diagnosis and medication events, respectively (i. e., labels I10, E92, Heparin, Mupirocin Ointment 2%, etc. in Figure 1).

Pre-processing. There is a large number of distinct ICD-10-CM diagnostic codes (and the analog ATC-codes for medications) in the MIMIC-IV data. Since the atrial fibrillation cohort is limited to 6535 patients, pre-processing is needed to reduce the number of node labels the algorithm is required to learn. The data was processed by: (1) Dropping all Condition nodes which correspond to earlier versions than ICD-10-CM (ICD-9 and a few ICD-8 codes). (2) Only keeping the chapter (the three first characters) of the ICD-10-CM

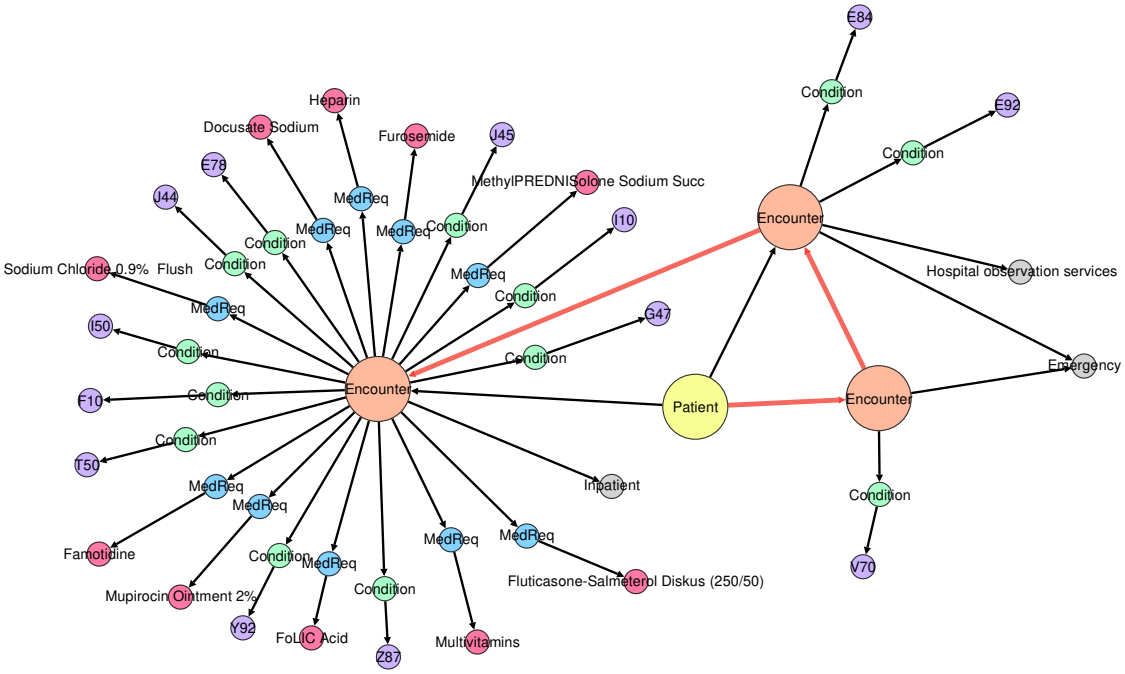


Figure 1: Visualization of a patient trajectory represented as a directed acyclic graph. The patient timeline is represented by the edges between Encounters (highlighted in red).

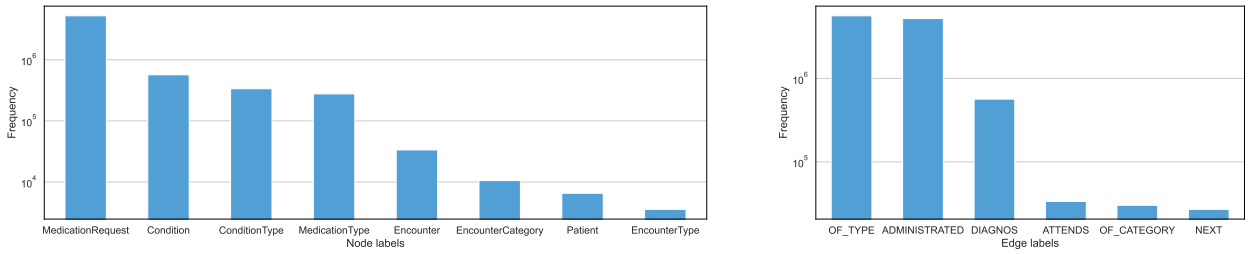


Figure 2: Histogram of the number of (higher level) node labels and edge labels in the training data.

codes, and merging nodes that ended up as identical. (3) Dropping rare events (condition and medication nodes that occurred less than 50 times). After these steps, 944 node labels remained. The largest graph had 143 Encounters and the largest Encounter had 180 successors. More details about the pre-processed trajectories are given in Table 1.

2.2 Generating model

Graph learning algorithms are usually permutation invariant to the ordering of nodes, and nodes are added sequentially, one at a time. Node ordering therefore becomes important. By modeling patient trajectories as DAGs, graph generation is significantly simplified, because every DAG has a at least one linear ordering of the nodes such that for every directed edge (u, v), node u comes before node v. This is known as a topological ordering and can be computed in linear time.

We found that standard recurrent neural networks were unable to learn realistic patient trajectories (see the discussion in the Methods section). We therefore designed a model tailored to the structure of our patient graphs. These trajectories are built up of linear sequences of Encounter nodes, where each Encounter node

	Raw	**After Pre-processing**
Max # nodes	18,947	2,772
Min # nodes	10	10
Average # nodes	1,044.1	221.2
Max # edges	36,811	5,162
Min # edges	9	9
Average # edges	1,867.3	294.7
# node labels ($\lvert\Sigma_V\rvert$)	13,980	944
# edge labels ($\lvert\Sigma_E\rvert$)	6	6
# graphs	6,535	6,535

Table 1: Statistics on the patient trajectories calculated from the atrial fibrillation cohort extracted from the MIMIC-IV database.

is the center of a star graph. These two substructures (linear sequence and star) can be modelled separately.

The new model is a variant of a variational autoencoder. The encoder maps patient trajectories into a parameterized multivariate Gaussian distribution (i. e., the encoder predicts the mean vector and covariance matrix of this distribution). A random sample is drawn from the distribution (the "hidden" representation of the input patient trajectory) and fed into the decoder to reconstruct the original patient trajectory. Once trained, new trajectories can be generated at scale by drawing random samples from a parameterized normal distribution and using the decoder to output a synthetic trajectory. Further details on the model architecture and training details are found in the Methods section.

2.3 Experiments

Graph reconstruction. We first investigated whether the proposed model can accurately reconstruct the input graphs. We found that the model achieved a lowest reconstruction loss at 0.02 (more details about the reconstuction loss function are given in the Methods section). We also used graph kernels to quantitatively measure the reconstruction loss. A graph kernel is a positive semi-definite kernel on the set of graphs $\mathcal{G}$ [26]. Roughly speaking, a graph kernel measures the similarity of graphs. Once we define a function $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ on the set $\mathcal{G}$, there exists a map $\phi : \mathcal{G} \to \mathcal{H}$ into a Hilbert space $\mathcal{H}$, such that $k(G, G') = \langle \phi(G), \phi(G') \rangle_{\mathcal{H}}$ for all $G, G' \in \mathcal{G}$ where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product in $\mathcal{H}$. Graph kernels are grouped into major families that focus on different structural aspects of graphs. We primarily relied on the Weisfeiler-Lehman subtree (WL) kernel [35] and on the shortest path (SP) kernel [7] to compare input graphs against reconstructed graphs. WL and SP are among the most successful graph kernels and account for both graph structure and node label information.

We computed the histogram of $k_i = k(G_i, \hat{G}_i)$, where G_i is an input graph, $\hat{G}_i$ its corresponding reconstructed graph, and $k(\cdot, \cdot)$ is a graph kernel (i. e., WL or SP) with $i \in \{1, 2, \ldots, 6535\}$. Here, $k_i = 0$ means that reconstructed graph i is completely different to its input, and vice versa $k_i = 1$ implies identity up to isomorphism. For the reconstruction task, ideally, we would like the model to output graphs isomorphic to those given as input. Thus, we would like most kernel values to be large (close to 1). The histogram in Figure 3 shows a very high similarity ($k > 0.9$ for 3/4 of the graph distribution) for most graphs. This indicates that the proposed model yields very good performance in reconstructing the input graphs even though some of them are relatively large and consist of several Encounter nodes.

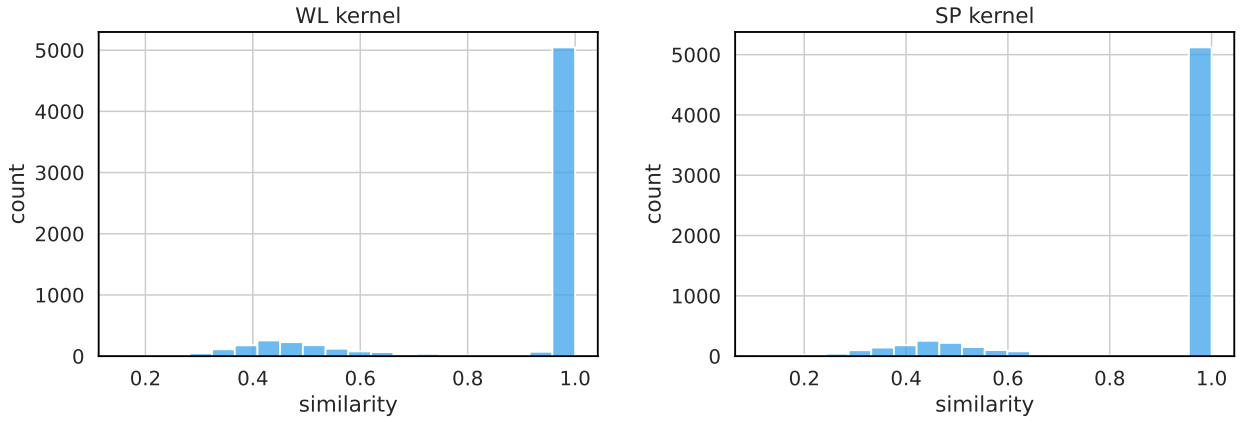


Figure 3: Histogram of similarities between input graphs and reconstructed graphs using the Weisfeiler-Lehman subtree (WL) kernel and the shortest path (SP) kernel.

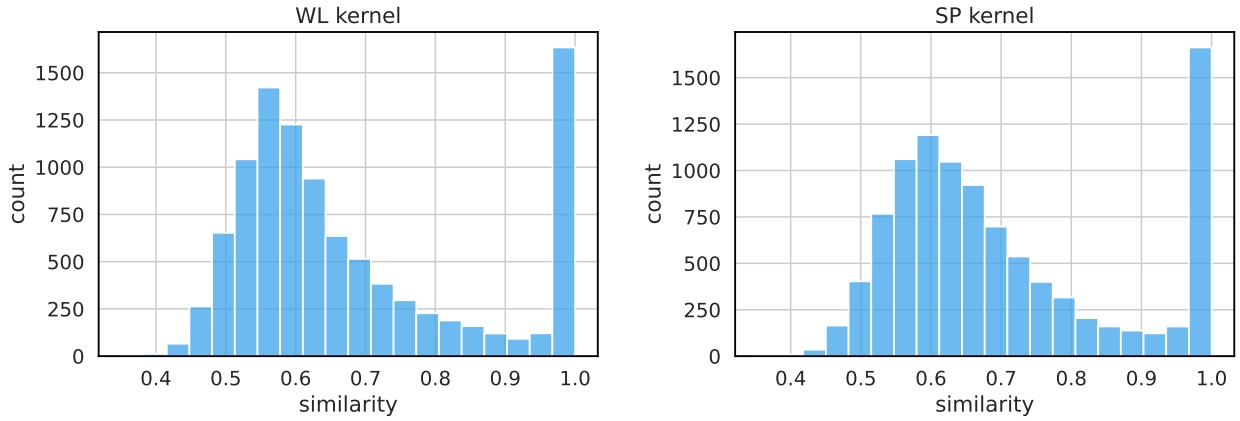


Figure 4: Similarity histogram between input and generated graphs using the Weisfeiler-Lehman subtree (WL) kernel and the shortest path (SP) kernel.

Graph generation. We have found that the model successfully reconstructs (learns) the patterns from the input graphs. Can the model also generate novel synthetic graphs that are realistic but not found in the training data? To investigate this, we generated 10000 synthetic graphs by feeding random samples drawn from the multivariate normal distribution to the decoder.

We first compared the generated synthetic graphs to the input graphs from the training data using graph kernels. Once again, we used the WL kernel and the SP kernel. We computed the histogram of the maximum similarity $k_j^{\max} = \max_i k_{ij}$ for the two kernels, where now G_i is an input graph, $\hat{G}_j$ is a graph generated from a random sample with $j \in \{1, 2, \ldots, 10000\}$, and $k_{ij} = k(G_i, \hat{G}_j)$ is an element of a (6535×10000)-similarity matrix with $k(\cdot, \cdot)$ being a graph kernel. Once all kernel values are computed, we end up with 10000 values for each kernel.

Figure 4 shows that the maximum similarity distributions are mainly centered around $k^{\max} = 0.5$ and $k^{\max} = 0.6$, regardless of kernel. There are also samples for which $k^{\max} \approx 1$ holds. Such graphs correspond to identical or nearly-identical copies of input graphs and could lead to patient privacy leaking from the training set. To reduce the risk of privacy leaking, these graphs need to be eliminated from the data set. Fortunately, the number of those graphs is not very large compared to the whole population of synthetic graphs. Thus, the set of generated graphs mainly consists of novel samples ($\approx 85\%$ of the whole population). Also note

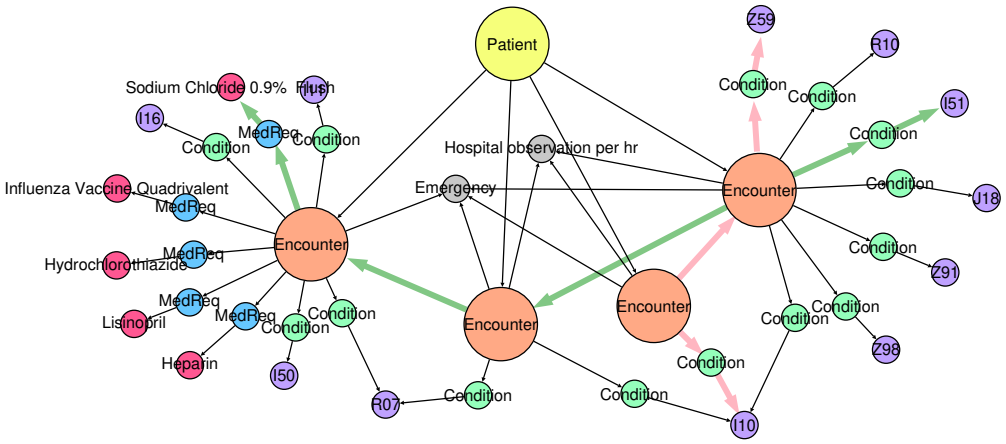


Figure 5: Examples of a 2-path and a 3-path in a patient trajectory. A 2-path from the ConditionType node I10 of the first Encounter node to ConditionType node Z59 of the second Encounter node is highlighted in pink, while a 3-path from the ConditionType node I51 of the second Encounter node to MedicationType node Sodium Chloride 0.9% of the fourth Encounter node is highlighted in green.

n	r	Description
1	0.991	node types
2	0.977	sequences of condition/medication codes of length two
3	0.952	sequences of condition/medication codes of length three

Table 2: Pearson's r between frequency of different structures in the collection of training graphs and the collection of generated graphs.

the low left wing contribution in the distributions. Contributions near $k^{\max} = 0$ would have indicated very low similarity to the inputs and generated patient trajectories that are unrealistic.

Further, we must determine to what extent these novel generated samples are realistic representations of electronic health records. In what follows, we remove samples that are very similar to input graphs (those graphs $\hat{G}_j$ for which $k_j^{\max} > 0.9$ according to the WL and/or SP kernel). We first investigate if paths of length n occurred with the same frequencies in the generated graphs as in the input graphs. Such n-paths can be thought of as Condition and Medication node pairs separated in time by $(n-1)$ Encounters. In this terminology, 1-paths correspond to node labels, 2-paths to nearest-neighbors, and 3-paths to next-nearest-neighbors. The first is a static (time-independent) property, but the other two are dynamic properties through the timeline implicit by the NEXT-relation between neighboring Encounters. A few examples of 2- and 3-paths are highlighted in Figure 5.

The number of such paths increase exponentially which has a significant impact on the time complexity to compute correlation coefficients for large n. We calculated Pearson's r as a function of n (Table 2 and Figure 6). The static 1-paths (node labels) are perfectly retained in the generated graphs ($r = 0.991$). This shows that static properties like patient attributes, and number of diagnosis and medications are indistinguishable in the generated graphs compared to the training data. The dynamic 2- and 3-paths are almost equally well preserved in the novel training samples ($r > 0.95$). This shows without a doubt that the model learns time-dependencies between conditions and medications that occur in consecutive Encounters.

We also validated our proposed model with a numerical experiment. We generated 6535 synthetic trajectories of a similar size (i. e., number of nodes) distribution to those of the training data. We first checked that no synthetic trajectory was isomorphic to any real trajectory. Then, we performed a graph classification

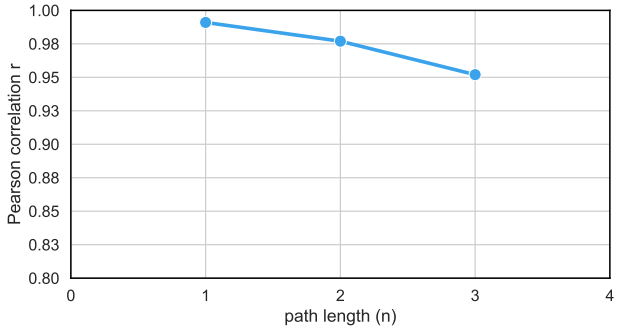


Figure 6: Pearson Correlation between frequency of different structures in the collection of training graphs and the collection of generated graphs.

	Without Labels	With Labels
WL	58%	72%
SP	56%	73%

Table 3: Classification accuracy of WL and SP kernels on the test data set (which consists of equal numbers of real and synthetic patient trajectories).

experiment to investigate whether a classifier can distinguish between real and synthetic trajectories using the WL and SP kernels. Specifically, the 13070 samples (real and synthetic data) were split with ratios $60:20:20$ into training, validation, and test sets. Then, the graph kernels were used to calculate similarity matrices that were passed to a Support Vector Machines (SVM) classifier [27]. Once trained, SVM can predict whether a test trajectory is real or synthetic. (We used the validation set to optimize the C-parameter of the SVM classifier). The classification accuracy is shown in Table 3. The accuracy measures the number of correctly classified test samples divided by the number of test samples. Here, in contrast to most classification problems, the goal is not to achieve a high accuracy (i. e., close to 100%), because that implies that the classifier can distinguish real from synthetic trajectories. Instead, a classification accuracy of 50% means that the classifier is no better than a random guess. We performed two different experiments. In the first one, we stripped node labels and just considered graph structure. The second experiment considered both structure and node labels. In terms of structure, the real and synthetic trajectories are very similar and both WL and SP kernels fail to predict whether a test trajectory is real or synthetic. On the other hand, we can see that including the information of the node labels helps the classifier's prediction. Considering the high correlations we found for n-paths, this indicates that some combinations of trajectory node labels are more frequent in synthetic graphs than their real counterparts, or vice versa.

Finally, it is enlightening to visualize a few novel patient trajectories that have been generated with this model, and compare them to some samples from the training set (Figure 7). It is clear that it would be very difficult for even the trained eye of a clinical specialist to distinguish a synthetic patient trajectory from a real one.

3 Discussion

Well generated synthetic healthcare data could provide an opportunity to improve the value of analytics by allowing easier access to data in order to pre-train AI models, generate novel hypotheses, and explore data patterns without jeopardizing patient's integrity. In this paper we present a deep learning model for generating synthetic patient trajectories from electronic health records. We show that the model can be

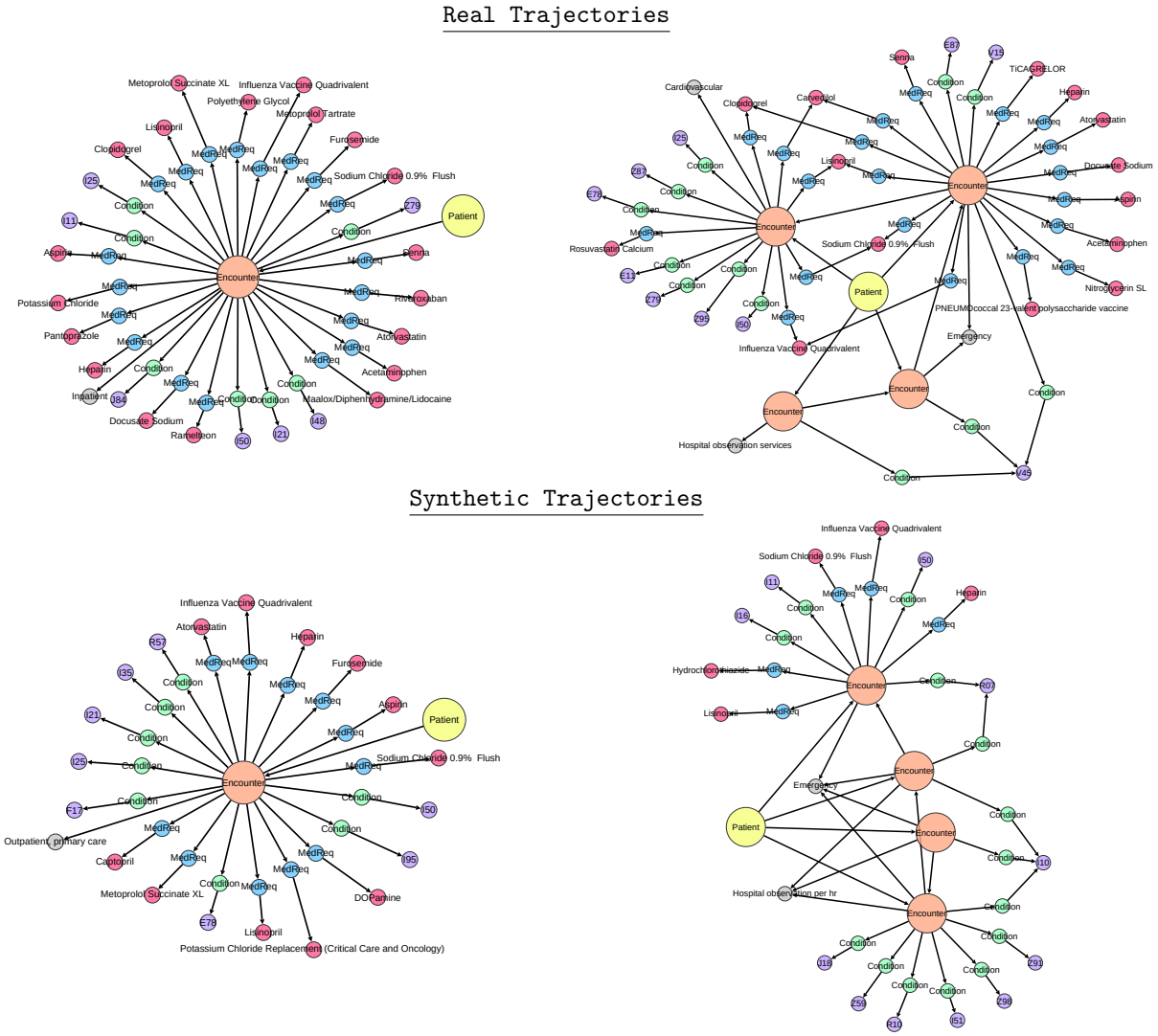


Figure 7: Visualizations of patient trajectories that were generated with the proposed model vs samples from the training data.

effectively trained on real graphs and generate novel ones, that are not in the training set. These patient trajectories are clinically realistic while sufficiently different from the trajectories in the training set to preserve patient privacy.

Our model is a Variational Graph Autoencoder (VGAE) tailored to generate patient trajectories represented as directed acyclic graphs. Previously existing generating models fail to produce large graphs or to learn long-range time correlations. The model proposed here solves these issues by decoupling the sequential patient timeline from the clinical interventions. The model is well suited for the complex time-dependencies found in electronic health records. Our numerical results show that the model generates novel synthetic patient trajectories, not found in the training data, that are sufficiently different to preserve patient privacy, yet retains the characteristics of the real-world data. Arguably the most significant feature is that the model is powerful enough to learn long-range correlations between trajectory nodes.

An interesting question rising from this work is to what extent synthetic data can replace real-world data in downstream analysis. Given our experimental results, and the model's ability to learn paths in patient

trajectories, we expect analysis based on either real or synthetic to lead to similar conclusions. This could be tested in practice by comparing output of machine learning classifiers trained on synthetic trajectories against those trained on real-world data. Are the data-driven insights from the two cases identical?

As we have already emphasized, the success of any deep learning model rests on the quality and amount of training data. The model can capture general trends already from limited training data, but ultimately requires large amounts of training data to generate long and accurate patient trajectories. In short, as always, the more data the better results. In general, outliers (patient trajectory groups that rarely occur in the training set) are also difficult to generate with accuracy. The generative model should always take measures to ensure that all trajectories of interest are well-represented in the training set.

Limitations. First, node and edge labels are the only metadata included in our model. There is a lot of additional metadata in EHR systems (for example, lab data including values and units) that is interesting for analysis. Such data is represented as key/value pairs on nodes and edges in our graph model. Our generator is easily extended to node and edge attributes by coupling a multi-layer perceptron (MLP) to the model once the node type has been determined. Second, the present version of the model assumes that Encounter nodes are connected to only one type of Condition or Medication node. In practice, there can be more than one node (if, for example, the patient is administered the same medication multiple times in the same encounter). This limitation is due to the model's binary classifier, which decides whether (or not) a single node of each type should be added to the encounter. A future iteration of the model could replace the binary classifier with a module which accounts for multiplicity. Third, the model has a number of hyper-parameters (see the Methods section) that could be investigated for further sensitivity analysis and optimization.

Privacy risks in the context of synthetic data. The greatest threat is if a malicious agent can use the synthetic patient trajectories to re-identify real patients from the training data. This is called privacy leaking. That risk is magnified when the agent is in possession of extra information about the real individuals (medical conditions, prescriptions, etc.) that can be combined with the synthetic data to form recognizable patterns that can be used for re-identification. In our model, the amount of similarity between the synthetic and real trajectories is adjustable by the amplitude of the noise injected into the sampled latent space. Synthetic data should undergo a careful evaluation with respect to identity disclosure risks prior to distribution [32]. A number of different approaches for reducing the risk of information disclosure [31, 28] has been proposed, since disclosure control methods have a significant impact on data utility.

Conclusion and perspectives. Graph deep learning is a powerful tool for learning complex data patterns. Here, a variant of a variational graph autoencoder (VGAE) tailored to generate patient trajectories represented as large directed acyclic graphs created privacy-preserving and highly accurate synthetic EHRs with long-range time correlations. This approach could reduce the problem of restricted access to health data, thus enabling explorative analyses, algorithm pre-training, hypothesis generation, and data expansion without jeopardizing privacy.

4 Methods

4.1 Notation

Let $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ for $n \geq 1$ and $G = (V, E)$ be a directed graph where V is the node set and E is the edge set, such that n is the number of nodes and m is the number of edges in the graph. The neighbourhood $\mathcal{N}(v)$ of a node v is the set of all nodes adjacent to v. For a directed graph, we use $\mathcal{N}^+(v) = \{u \,|\, (v, u) \in E\}$ to indicate the set of out-neighbors of v where (v, u) is an edge between nodes v and u of V, and $\mathcal{N}^-(v) = \{u \,|\, (u, v) \in E\}$ to indicate the set of in-neighbors of v. The out-degree of node v is $d^+(v) = |\mathcal{N}^+(v)|$ and its in-degree is $d^-(v) = |\mathcal{N}^-(v)|$. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ of a graph G is a symmetric (and typically sparse) matrix used to encode edge information in the graph. Element (i, j) is the weight of the edge between nodes v_i and v_j if the edge exists and 0 otherwise. For graphs with node labels and edge labels, nodes and

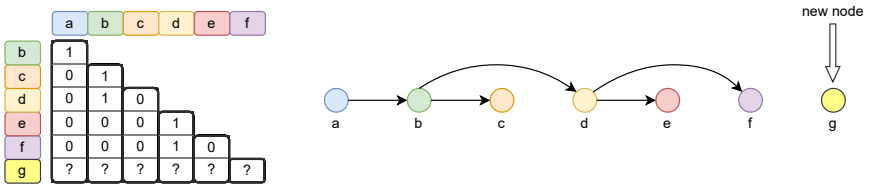


Figure 8: Approach that models graphs in an autoregressive manner.

edges are associated with discrete labels, expressed by two functions $\ell_V : V \to \Sigma_V$ and $\ell_E : E \to \Sigma_E$ that map nodes and edges to labels from the sets of labels Σ_V and Σ_E, respectively.

4.2 A model tailored to patient trajectories

The direct way to generate patient trajectories is to add nodes and edges auto-regressively (as a sequence) from a topological order with a permutation π. Since patient trajectories are DAGs, it is enough to generate the lower triangular part of the adjacency matrix (Figure 8). For each new node that is generated, the model needs to decide whether this node is connected to each of the previously generated nodes. This corresponds to $n(n-1)/2$ probabilities for a graph with n nodes. This is impractical since patient trajectories are graphs with $n > 100$.

We also found that the recurrent architectures including Gated Recurrent Units (GRUs) [11] and Long Short-Term Memory layers (LSTMs) [17] can not learn long-range interactions between nodes. To realize this, let v denote an Encounter node and $u_1, u_2, \ldots, u_r$ are its successors of Condition and MedicationRequest types. All these nodes come after v in the topological order but they all depend on v. It is very difficult for a recurrent layer to capture the interaction between v and nodes that are very far from v in the ordering when r is large.

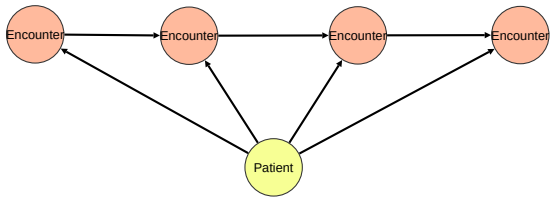
We designed a model tailored to patient trajectories to solve these issues. In this model, each graph corresponds to:

- A Patient node followed by a sequence of Encounter nodes (Figure 9a).
- Each Encounter node is connected to Condition and MedicationRequest nodes, which in turn are terminated with ConditionType and MedicationType nodes. An Encounter node could also be connected to EncounterType and/or EncounterCategory nodes (Figure 9b).

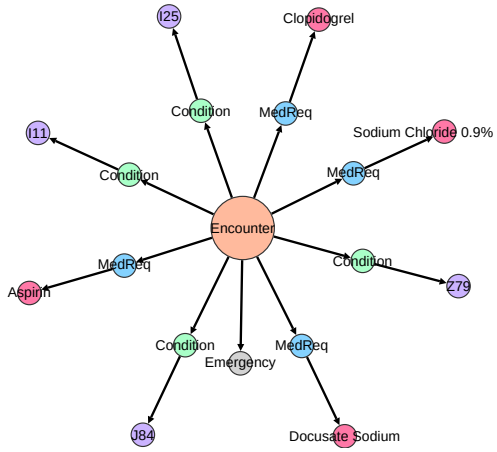
Clearly, the graph generation can be carried out in two steps: (1) Generate the Encounter node sequence. (2) Generate the successors of each Encounter node. For the first task, we use the topological order of the patient trajectory subgraph obtained only from the Patient and Encounter nodes. This topological order is important because it keeps the trajectory timeline by enforcing Encounter node u to precede Encounter v chronologically. Since Patient and Encounter nodes are only a small fraction of the nodes in the patient trajectory, a recurrent neural network (RNN) can capture the relationships between consecutive encounters in the sequence. For the second task, we could generate successors of the Encounter nodes by imposing any topological ordering and let another RNN learn that structure. That is possible since Encounter nodes do not have too many successors. In this work, we used an alternative approach where we consider the Encounter successor nodes as a set, and then we simply generate a set that contains those nodes.

4.3 Architectural details

We use an encoder-decoder architecture. The encoder maps input DAGs to a distribution parameterized as a multivariate Gaussian. In other words, the encoder predicts the mean and standard deviation of this Gaussian distribution. A random sample is then drawn from the distribution and serves as the latent representation



(a) Example of a sequence of 4 Encounter nodes.



(b) Example of an Encounter node.

Figure 9: The two main structures into which a patient trajectory can be decomposed: (a) a Patient node followed by a sequence of Encounter nodes; and (b) a set of Encounter nodes, each connected to several Condition and MedicationRequest nodes, which in turn are terminated with ConditionType and MedicationType nodes.

of the input graph. The decoder tries to reconstruct the input DAGs given their vector representations. The decoder is a variational approximation, $p_\theta(G|\mathbf{z})$, which takes an embedding $\mathbf{z}$ as input.

Two pre-processing steps were applied to the patient trajectories before encoding. First, we merged Condition/ConditionType and MedicationRequest/Medication type node pairs. Second, for each graph, an End node was added via a directed edge to the last Encounter node. This allows the model to decide when to terminate the generation of nodes in a new graph.

Encoder. The encoder of the model is a message passing graph neural network. Its first part is an embedding layer that creates representations for the nodes in each patient DAG. Each node v has a trainable node embedding $\mathbf{x}_v$, and there is a single node embedding for each node type. These node embeddings are updated during training with a combination of synchronous and asynchronous message passing schemes.

First, the Encounter node embeddings are updated by aggregating the embeddings of their successors, excluding Encounter and End nodes:

$$\begin{aligned} \mathbf{m}_v &= \sum_{u \in \mathcal{N}^+(v)} f(\mathbf{x}_u) \\ \mathbf{h}_v &= \mathrm{GRU}(\mathbf{x}_v, \mathbf{m}_v) \end{aligned} \tag{1}$$

where $\mathcal{N}^+(v)$ is the set of successors of Encounter node v (again, excluding Encounter and End nodes), f is a neural network (MLP), $\mathbf{x}_v$ is the embedding of node v, and GRU is a gated recurrent unit.

An asynchronous message passing scheme is then applied where we sequentially perform message passing according to the topological sorting obtained from the patient subgraph of Encounter and End nodes. This differs from the standard message passing scheme in graph neural networks where all node embeddings are updated at each algorithm step. In our algorithm, the node embeddings are updated in this step according to:

$$\begin{aligned} \mathbf{m}_v &= \sum_{u \in \mathcal{N}^-(v)} f(\mathbf{h}_u) \\ \mathbf{h}_v &= \mathrm{GRU}(\mathbf{h}_v, \mathbf{m}_v) \end{aligned} \tag{2}$$

where $\mathcal{N}^-(v)$ is the set of incoming neighbors of v for Encounter nodes.

Once all node embeddings of the DAG have been computed, we use the end node embedding (i. e., the node without any successors) as the output of the encoder. Thus, $\mathbf{h}_G = \mathbf{h}_e$ where e denotes the End node of G. This vector is passed to two fully-connected layers to get the mean and variance parameters of the posterior approximation $q(\mathbf{z}|G)$:

$$\begin{aligned} \boldsymbol{\mu} &= \mathbf{W}_{\boldsymbol{\mu}}\mathbf{h}_G + \mathbf{b}_{\boldsymbol{\mu}} \\ \log \boldsymbol{\sigma}^2 &= \mathbf{W}_{\boldsymbol{\sigma}^2}\mathbf{h}_G + \mathbf{b}_{\boldsymbol{\sigma}^2} \end{aligned} \tag{3}$$

Decoder. The decoder of the model also applies an asynchronous message passing scheme to generate node representations. The decoder uses a GRU to update node embeddings when generating the graph.

A fully-connected layer is used to map the input latent vector $\mathbf{z}$ to the initial (hidden) state vector $\mathbf{h}_0$. The state vector is passed to the GRU, which constructs a DAG node-by-node. So far, all are Encounter (or End) nodes. The embedding of the first (Patient) node is $\mathbf{h}_{v_1} = \mathrm{GRU}(\mathbf{x}_{v_1}, \mathbf{h}_0)$. The following steps are performed to generate node v_i:

1. Compute the label distribution of v_i with an MLP based on the current graph state $\mathbf{h}_G = \mathbf{h}_{v_{i-1}}$.
2. Sample the label of v_i. If this is the end label, stop the decoding, connect the last Encounter node to v_i, and return the DAG. If not, continue the generation.
3. Connect the last added Encounter node and the Patient node to v_i. Update $\mathbf{h}_{v_i}$ according to:

$$\begin{aligned} \mathbf{m}_{v_i} &= \sum_{u \in \mathcal{N}^-(v_i)} f(\mathbf{h}_u) \\ \mathbf{h}_{v_i} &= \mathrm{GRU}(\mathbf{x}_{v_i}, \mathbf{m}_{v_i}) \end{aligned} \tag{4}$$

4. Produce a vector $\mathbf{s} \in \mathbb{R}^c$ (c denotes the different types of successors of Encounter nodes excluding Encounter and End nodes):

$$\mathbf{s} = \mathrm{MLP}(\mathbf{h}_v) \tag{5}$$

   The sigmoid function is applied point-wise to the MLP output and then the model decides whether to add a node of each type of successor to the graph. When a new node is added, so is a directed edge from v_i. The decision to add a successor to the graph is a binary classification problem. We therefore use the binary cross entropy loss to train the model.

4.4 Loss function

The loss function of our variational autoencoder has two terms,

$$\mathcal{L} = \mathcal{L}_{\text{reconstruction}} + \mathcal{L}_{KL} \tag{6}$$

The first term is the reconstruction loss, i. e., the variational lower bound, and measures how well the model reconstructs the input data. The reconstruction loss is high if the reconstructed DAG is very different from its input. This term can be split into two contributions, $\mathcal{L}_{\text{reconstruction}} = \mathcal{L}_{\text{encounter}} + \mathcal{L}_{\text{other}}$. One contribution measures how well the model can reconstruct the sequence of Encounter nodes. It is equal to the binary cross-entropy between the predicted types of Encounter or End nodes and their actual types:

$$\mathcal{L}_{\text{encounter}} = -\frac{1}{N}\sum_{i=1}^{k} \ell(v_i)\log(\hat{y}_i) + (1-\ell(v_i))\log(1-\hat{y}_i) \tag{7}$$

Remember, $\{v_1, \ldots, v_k\}$ are only the Encounter and End nodes in the DAG. The other contribution measures how well the model can reconstruct the successors of the Encounter nodes. It is equal to the binary cross-entropy between the predicted and the actual successors of each Encounter node:

$$\mathcal{L}_{\text{other}} = -\frac{1}{N} \sum_{i=1}^{k} \sum_{j=1}^{r} f(v_i, \sigma_j) \log(\hat{y}_{ij}) + (1 - f(v_i, \sigma_j)) \log(1 - \hat{y}_{ij}) \tag{8}$$

Here the $\{\sigma_1, \ldots, \sigma_r\}$ set includes all nodes except Encounter and End nodes for node v:

$$f(v, \sigma) = \begin{cases} 1, & \left|\{u | u \in \mathcal{N}^+(v) \wedge \ell(u) = \sigma\}\right| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

The second term of the loss function is a regularization term. It is equal to the Kullback–Leibler (KL) divergence of the approximate $q(\mathbf{z}|G)$ from the true posterior $p(\mathbf{z})$, where $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{0}$ and $\mathbf{I}$ are the all-zeros vector and the identity matrix, respectively. The KL divergence measures how closely the output distribution $q(\mathbf{z}|G)$ matches $p(\mathbf{z})$:

$$\mathcal{L}_{KL} = -\text{KL}[q(\mathbf{z}|G)||p(\mathbf{z})] \tag{10}$$

4.5 Experimental Setup

We used the following values for the model's hyper-parameters. The hidden-dimension size of the embedding layer and the GRU layers were 512. The hidden-dimension size of the fully-connected layer that transforms the sampled vector representation of the graphs was set to 512 and followed by a tanh-activation. We used an MLP with hidden-dimension size 1024 to decide whether a new node type was to be added to the graph (and also to determine its type). We used an MLP with hidden-dimension size 2048 to compute the successors of Encounter nodes. The hidden layers in both MLPs were followed by ReLU activation functions. The dimension of the multivariate Gaussian distribution was set to 256. The batch size was 256 and the number of learning epochs was 5000. We used the Adam optimizer with an initial learning rate of 10^{-3} and decayed the learning rate by 0.1 every 1000 epochs to a minimum of 10^{-5}. The model with the lowest training loss was stored on disk and retrieved at the end of training. The best model was then used to generate new graphs for the numerical experiments.

Data Availability

MIMIC-IV data is available on the PhysioNet repository (`https://physionet.org/`) and access is authorized to users through a data use agreement with the providers.

Acknowledgements

M.V. is partially supported by the "Wallenberg AI, Autonomous Systems and Software Program" (WASP). M.L. is partially supported by AIR Lund (Artificially Intelligent use of Registers at Lund University) research environment, and received funding from the Swedish Research Council (VR; grant no. 2019-00198). G.N. is supported by the French National research agency via the AML-HELAS (ANR-19-CHIA-0020) project.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[2] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys*, 51(4):1–35, 2018.

[3] Theodoros N Arvanitis, Sean White, Stuart Harrison, Rupert Chaplin, and George Despotou. A method for machine learning generation of realistic synthetic datasets for validating healthcare applications. *Health Informatics Journal*, 28(2), 2022.

[4] Mrinal Kanti Baowaly, Chia-Ching Lin, Chao-Lin Liu, and Kuan-Ta Chen. Synthesizing electronic health records using improved generative adversarial networks. *Journal of the American Medical Informatics Association*, 26(3):228–241, 2019.

[5] Duane Bender and Kamran Sartipi. Hl7 fhir: An agile and restful approach to healthcare information exchange. In *Proceedings of the 26th IEEE International Symposium on Computer-Based Medical Systems*, pages 326–331, 2013.

[6] Pietro Bongini, Monica Bianchini, and Franco Scarselli. Molecular generative graph neural networks for drug discovery. *Neurocomputing*, 450:242–252, 2021.

[7] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining*, 2005.

[8] Michail Chatzianastasis, George Dasoulas, Georgios Siolas, and Michalis Vazirgiannis. Graph-based neural architecture search with operation embeddings. In *Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision Workshops*, pages 393–402, 2021.

[9] Richard J Chen, Ming Y Lu, Tiffany Y Chen, Drew FK Williamson, and Faisal Mahmood. Synthetic data in machine learning for medicine and healthcare. *Nature Biomedical Engineering*, 5(6):493–497, 2021.

[10] Kieran Chin-Cheong, Thomas Sutter, and Julia E Vogt. Generation of heterogeneous synthetic electronic health records using gans. In *Workshop on Machine Learning for Health (ML4H) at the 33rd Conference on Neural Information Processing Systems*, 2019.

[11] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of 8th Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, 2014.

[12] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. In *Proceedings of Machine Learning for Healthcare 2017*, pages 286–305, 2017.

[13] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[14] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[15] Andre Goncalves, Priyadip Ray, Braden Soper, Jennifer Stevens, Linda Coyle, and Ana Paula Sales. Generation and evaluation of synthetic patient data. *BMC medical research methodology*, 20(1):1–40, 2020.

[16] Jie Gui, Zhenan Sun, Yonggang Wen, Dacheng Tao, and Jieping Ye. A review on generative adversarial networks: Algorithms, theory, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[18] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2323–2332, 2018.

[19] Alistair Johnson, Lucas Bulgarelli, Tom Pollard, Steven Horng, Leo Anthony Celi, and Roger Mark. Mimic-iv, 2021.

[20] James Jordon, Jinsung Yoon, and Mihaela Van Der Schaar. Pate-gan: Generating synthetic data with differential privacy guarantees. In *7th International Conference on Learning Representations*, 2019.

[21] Dhamanpreet Kaur, Matthew Sobiesk, Shubham Patil, Jin Liu, Puran Bhagat, Amar Gupta, and Natasha Markuzon. Application of bayesian networks to generate synthetic health data. *Journal of the American Medical Informatics Association*, 28(4):801–811, 2021.

[22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations*, 2014.

[23] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[24] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):712–733, 2020.

[25] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[26] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *Journal of Artificial Intelligence Research*, 72:943–1027, 2021.

[27] William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[28] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. *Proceedings of the VLDB Endowment*, 11(10), 2018.

[29] Martin Popel, Marketa Tomkova, Jakub Tomek, Łukasz Kaiser, Jakob Uszkoreit, Ondřej Bojar, and Zdeněk Žabokrtskỳ. Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals. *Nature communications*, 11(1):1–15, 2020.

[30] Giorgia Ramponi, Pavlos Protopapas, Marco Brambilla, and Ryan Janssen. T-cgan: Conditional generative adversarial network for data augmentation in noisy time series with irregular sampling. *arXiv preprint arXiv:1811.08295*, 2018.

[31] Jerome P Reiter. Satisfying disclosure restrictions with synthetic data sets. *Journal of Official Statistics*, 18(4):531, 2002.

[32] Jerome P Reiter and Robin Mitra. Estimating risks of identification disclosure in partially synthetic data. *Journal of Privacy and Confidentiality*, 1(1), 2009.

[33] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *npj Digital Medicine*, 3(1):1–7, 2020.

[34] Guillaume Salha, Stratis Limnios, Romain Hennequin, Viet-Anh Tran, and Michalis Vazirgiannis. Gravity-inspired graph autoencoders for directed link prediction. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 589–598, 2019.

[35] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[36] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Proceedings of the 27th International Conference on Artificial Neural Networks*, pages 412–422, 2018.

[37] Allan Tucker, Zhenchen Wang, Ylenia Rotalinti, and Puja Myles. Generating high-fidelity synthetic patient data for assessing machine learning healthcare software. *npj Digital Medicine*, 3(1):1–13, 2020.

[38] Jason Walonoski, Mark Kramer, Joseph Nichols, Andre Quina, Chris Moesel, Dylan Hall, Carlton Duffett, Kudakwashe Dube, Thomas Gallagher, and Scott McLachlan. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. *Journal of the American Medical Informatics Association*, 25(3):230–238, 2018.

[39] W Patrick Walters and Regina Barzilay. Applications of deep learning in molecule generation and molecular property prediction. *Accounts of chemical research*, 54(2):263–270, 2020.

[40] Philipp Wendland, Colin Birkenbihl, Marc Gomez-Freixa, Meemansa Sood, Maik Kschischo, and Holger Fröhlich. Generation of realistic synthetic data using multimodal neural ordinary differential equations. *npj Digital Medicine*, 5(1):1–10, 2022.

[41] Andrew Yale, Saloni Dash, Ritik Dutta, Isabelle Guyon, Adrien Pavao, and Kristin P Bennett. Generation and evaluation of privacy preserving synthetic health data. *Neurocomputing*, 416:244–255, 2020.

[42] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. Time-series generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2019.

[43] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5708–5717, 2018.