CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Off-policy latent variable modeling for fast bandit personalization

Master's thesis in Computer science and engineering

Ludvig Liljeqvist
Viktor Truvé

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Off-policy latent variable modeling for fast bandit personalization

Ludvig Liljeqvist
Viktor Truvé

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Off-policy latent variable modeling for fast bandit personalization

Ludvig Liljeqvist
Viktor Truvé

Supervisor: Fredrik Johansson, Department of Computer Science and Engineering
Examiner: Moa Johansson, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Off-policy latent variable modeling for fast bandit personalization

Ludvig Liljeqvist
Viktor Truvé
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Medical treatments are decided based on medical history and the current symptoms of a patient. For chronic illnesses this can be difficult, as long-time patients develop an amount of medical data that is hard to grasp. We propose the use of machine-learning methods to both condense this information, and then utilize it to recommend medical treatments. Our goal is thus to develop an efficient method for finding optimal treatments for patients – optimized for doing this in as few rounds of treatment as possible.
We do this in a two step process: the first step is to develop a generalist model for treatment recommendation using a combination of a seq2seq model, and a Variational Autoencoder (VAE). The VAE condenses intricate patient information into an encoding, and has the ability to reconstruct that information using this encoding. We can thus consider each possible encoding as a patient type, that indicates which treatment is best for that particular type, on average. Seq2seq adapts the VAE to be applicable to sequential data – in our case, medical records. The second step is to use the generalist model to produce specialized policies for individual patients, inside a latent bandit model.
The ambition is that this solution will lead to faster personalization compared to simpler methods, such as contextual bandits and multi-armed bandits, among others. We present results showing that the proposed model performs better in earlier rounds of treatment than other bandit algorithms, and also converges to a near-optimal policy faster.

Keywords: Machine Learning, Health Care, AI, Latent Variable Models, Multi-armed Bandits, VAE, Latent Bandit.

Acknowledgements

Special thanks to our supervisor, Fredrik Johansson, for his considerable guidance in both shaping this project, and helping it reach its conclusion. Additional thanks go to Newton Mwai Kinyanjui for providing the means to generate the data necessary for our solution.

Ludvig Liljeqvist and Viktor Truvé, Gothenburg, June 2022

Contents

List of Figures

1

Introduction

In the health care domain, treatment is typically recommended based on a patient's personal history, medical records, and symptoms. Patients with chronic illnesses go through many rounds of treatment during their lives, and it may take a medical professional many attempts before they find the best treatment.

Medical professionals combine their knowledge of what previous research indicates, alongside specific knowledge gathered from examining the individual at hand to suggest treatment. This practice is known as clinical decision-making. Unfortunately there still exist cases where information gathered this way is insufficient for recommending the correct treatment. For a chronic illness like rheumatoid arthritis, deciding the first treatment is simple; if that one fails, deciding the second is feasible. From then on however, the evidence for choosing treatment grows sparse [Solomon et al., 2021].

There could be many reasons why deciding a treatment is difficult. Besides the issue where not enough information about a patient is available, there is also the possibility that the available patient data is too extensive for a medical professional to analyze effectively. Finally, there may exist unobserved information in that data which also impacts the treatment outcome. Hidden information such as genotypes and other biomarkers can have recognizable effects on the prognoses of patients – to such a degree that subtypes of patients within a disease can be identified [De Jager, 2009, Planey and Gevaert, 2016].

The normal practice in health care therefore relies on the assumption that previous observations and the current observation tell the professional everything they need to know to perfectly treat patients. For problems of this category, one could utilize offline reinforcement learning on historical data to produce a general model for patient treatment. The utilization of reinforcement learning in the health-care domain has previously been used in many different areas [Tseng et al., 2017, Raghu et al., 2017, Zhao et al., 2011, Guez et al., 2008, Chakraborty and Moodie, 2013]. With such a model, patient treatment can simply be recommended based on a patient's similarity to previously observed patients. This method aims to produce somewhat of a one-fits-all kind of model, assuming that whatever uniqueness observed in a patient does not hinder their optimal treatment from being the same as that for patients with similar traits.

Conversely, one may consider an online reinforcement learning approach such as a bandit algorithm [Lattimore and Szepesvári, 2020, Tennenholtz et al., 2021, Hong et al., 2020, Slivkins, 2019]. In this instance one would make no assumptions about other patients, and learn a personalized policy of treatment sequences from scratch for each one. Furthermore, there exist versions of bandit algorithms which can work with unobserved variables. Models which cater to this sort of problems are referred to as Latent Variable Models (LVMs) [Abbi et al., 2008, Saunders et al., 2020, Kingma and Welling, 2014, Sachdeva et al., 2019]. LVMs tackle the aforementioned fact that important information needed for treatment suggestion may be unobserved (latent). A benefit of combining such a model with a bandit algorithm is that bandit algorithms are guaranteed to converge to an optimal policy under the right circumstances [Lattimore and Szepesvári, 2020]. The problem with using a bandit approach is that it would require performing thousands of trials on the patient before an optimal policy is reached. This is incredibly dangerous, and not feasible.

Given that both of these approaches have shortcomings which make them unfit for the problem we are addressing, a new method is needed. In the paper Håkansson et al. [2020], a method for searching for near-optimal treatment methods for patients is presented. This approach utilizes historical data to perform policy optimization to find near-optimal treatments in as few trials as possible. In Håkansson et al. [2020], a distribution of different treatment outcomes given the historical observations is estimated from the data. Furthermore, Håkansson et al. assume that the possible historical observations can be modelled by a number of discrete states. Given that the number of different possible observable histories grows exponentially with each treatment round, the problem of accounting for them all is NP-hard. In our approach we let this distribution remain unknown, and instead we train a model from scratch on historical data, allowing the model to learn an estimate of the outcome distribution on its own. This estimate is then used to predict which sequence of treatments corresponds to the best outcomes.

In this paper we present a combination of offline learning on historical data to achieve a general model, together with a bandit algorithm for online learning to achieve a personalized policy. More specifically we propose an LVM that combines the logic within a seq2seq neural network [Sutskever et al., 2014] with that of a variational autoencoder [Kingma and Welling, 2014] for offline learning, and a latent bandit model for online learning. The LVM identifies subtypes in a population of patients, and learns to estimate the average outcome of treatments based on this subtype. For each patient, the latent bandit then administers treatment according to the specification of the LVM, and solidifies its belief of treatment outcome and subtype for that individual.

We evaluate our work by answering the following research questions:

1. How close is the bandit's policy to an optimal policy? / Does the bandit learn an optimal policy?

2. How well does the LVM encapsulate patient types?

3. Do we achieve faster personalization?

The architecture of our offline learning model, as well as the bandit is further described in section 3.2 and the empirical study of results is presented in chapter 4. It is worth noting that, as we limit ourselves to data generated from a simulator it is uncertain how our model would perform in a real-world setting. We do however still believe there are interesting insights to be gained by studying the results and performance of our solution.

2

Background

That statistical models can be used to optimize decision-making in health care is an idea that has been around for some time. In Schaefer et al. [2005] it has been demonstrated that Markov decision processes (MDPs) are a powerful technique for finding optimal policies in stochastic decision-making. Although the lack of electronic patient records has previously hampered the potential of MDPs in health care settings, this is no longer the case. Today, medical data is abundant, with data sets like GenBank and caBig [Benson et al., 2012, Fenstermacher et al., 2006]. This has allowed the use of MDPs, AI, and other statistical approaches to flourish.

The following chapter describes underlying concepts of our solution: section 2.1 gives a brief introduction to different bandit algorithms [Slivkins, 2019, Lattimore and Szepesvári, 2020]; section 2.2 describes the concepts relevant to latent variable modeling; section 2.3 describes sequence-to-sequence modeling – recurrent neural networks that turn sequential data into another kind of sequential data; lastly, section 2.4 describes the Alzheimer's Disease Causal Estimation Benchmark, a simulator of clinical variables associated with Alzheimer's disease.

2.1 Bandit Algorithms

A well-known reinforcement learning problem is the *Multi-armed bandit*. The Multi-armed bandit is a problem in which a fixed limited set of resources must be allocated between competing choices in a way that maximizes their expected gain. Usually, the properties of each choice are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the choice [Gittins et al., 2011, Katehakis and Veinott Jr, 1987].

The typical case for bandit algorithms is for them to be employed in online learning. This means that they act upon data as it is generated. In the setting of medical treatment recommendation, this translates to the bandit observing the current state of the patient, and deciding a treatment that they receive. This prompts the generation of the next state of the patient. Bandit algorithms are also fully applicable to offline learning. Rather than acting upon data as it arrives, in offline learning the bandit has access to a set of static data. It is prompted to act upon the states present in this data, and depending on the decision made, the next states are fetched from this set. In medical treatment recommendation – when a bandit employs offline

learning – rather than making decisions for a patient in real time, the bandit does so for historical patient data.

To put it simply, the most basic bandit algorithm has a fixed set of *arms*, i.e available actions A. The task of the algorithm is to for each round $t < T$ – where T is the number of rounds performed, fixed or otherwise – choose one action $a \in A$, "play" that action, and receive some reward R_t. The most basic form of bandit models are those which consider independent identically distributed (IID) rewards; these are referred to as stochastic bandits. In this case we assume that there exists a distribution D_a for each action such that every time an action is played, the reward is sampled – independently of previous rewards – from this distribution, $R_t \sim D_a$. The goal of the bandit is to learn the distribution D_a, if this is successfully done the bandit can effectively choose the action in each round which maximizes its expected reward $\mathbb{E}[\sum_{t=1}^{T} R_t]$.

The question which begs to be answered is how do we select actions a such that we can ensure that we do in fact learn D_a? For this purpose there exist many different algorithms such as explore-first, Epsilon-greedy, UCB, Thompson sampling, and more. How a bandit chooses an action is referred to as the bandit's policy, often denoted π. Then, the policy that yields the highest expected reward, is known as the optimal policy, denoted π^*. One of the possible problems with bandits is that depending on the policy, a great number of rounds T might be required to effectively learn the optimal policy.

In our setting we consider each patient as a separate case of the bandit-problem; we wish to train a new bandit for each patient. Given that our actions correspond to treatment methods, our goal is to find π^* or at least $\hat{\pi}^* \approx \pi^*$ in much fewer rounds than what is common for a bandit, since we can not excessively try treatment methods on patients.

2.1.1 Contextual bandits

A generalization of the problem above is the *contextual multi-armed bandit*, which also has information about a d-dimensional feature vector X that can aid decision making. In this case, the goal of the learner is to understand how the contexts and rewards correlate with one another, and then to use this information to perform the best actions. In the context of our work, the feature vector corresponds to the medical history of a patient, which we want to relate to treatment outcome.

2.1.2 Thompson Sampling

Thompson sampling [Thompson, 1933] is an algorithm intended to tackle the exploration-exploitation trade-off in multi-armed bandit problems. This trade-off essentially explains to what degree the bandit should perform the action it currently considers optimal (exploit) or whether to further investigate the value of some other action (explore).

In Thompson sampling we consider the same situation as that of a contextual bandit,

where we have a set of actions A, a set of contexts X, and a set of rewards R. In each round the algorithm receives a context $x \in X$ and chooses an action $a \in A$ and receives a reward $r \in R$. The elements of Thompson sampling consists of a likelihood function $P(r \mid \theta, a, x,)$ a set Θ of parameters θ over the distribution of r. A prior distribution P(θ) over these parameters, a history of previously observed values in the form $H = \{(x, a, r)\}$, and finally a posterior distribution $P(\theta \mid H) \propto P(H \mid \theta)P(\theta)$. In each round the algorithm samples parameters θ^* from the posterior $P(\theta \mid H)$ and chooses an action a^* to maximize the expected reward given $\mathbb{E}[r \mid \theta^*, a^*, x]$. Thompson sampling relates to our work in that it can be used to continuously reestimate the perception of a patient's subtype. Here, subtype takes the place of the parameters θ.

While Thompson sampling enjoys strong theoretical guarantees and worst-case sample complexity matching lower bound on regret, practical implementations typically require hundreds of samples to identify optimal actions. To improve this further, we can impose structural assumptions on the data generating process, which limits the number and nature of parameters we must learn for each patient.

2.1.3 Latent Bandits

The *Latent Bandit* is a form of multi-armed bandit which was invented to tackle the issue where the state information (context) is not enough to determine an optimal action. Instead, we need to find a way to discover the latent (unknown) information, which we then use to determine the optimal action. Especially relevant for our project is the *Model Thompson Sampling* (*mTS*) algorithm presented in Hong et al. [2020]. We here assume that we have a known and fixed model $\hat{\theta}$ such that $\hat{\theta} = \theta^*$. where θ^* is the optimal model. We also have a prior distribution P_1 over every latent state Z. What is then utilized is a variant of the Thompson Sampling algorithm called mTS:

Algorithm 1 mTS

Input:
Model Parameters $\hat{\theta}$
Prior over latent type $P_1(Z)$
for t $\leftarrow 1, \ldots, T$ **do**
  $P_t(Z) \propto P_1(Z) \prod_{\ell=1}^{t-1} P(R_\ell \mid A_\ell, X_\ell, Z; \hat{\theta})$
  Sample $Z' \sim P_t$
  Select $A_t \leftarrow \text{argmax}_{a \in A} \hat{\mu}(a, X_t, Z')$
end for

The algorithm works by maintaining a posterior probability $P_t = \mathbb{P}(Z_* = Z \mid H_t)$ which represents the probability of a given latent state Z being the optimal state Z_*. In our work, the latent state Z is conceptualized as the subtype of a patient. For instance, in a patient with Alzheimer's disease, Z can represent the level of amyloid plaques in the brain. As amyloid plaques have a discernible effect on a patient's prognosis and drug responsiveness [Kinyanjui and Johansson, 2021], it is

one of possibly several viable criteria of subtype. mTS considers a context that is time-varying, denoted by X_t. $\hat{\mu}$ denotes the estimated average reward from choosing an action a for a subject that has context X_t, and currently estimated latent type Z'.

2.2 Latent Variable Models

Latent Variable Models (LVMs) have previously been used for medical treatment prediction [Saunders et al., 2020]. An LVM is a probabilistic model that relates a set of observable variables – called *manifest variables* – to a set of latent variables. The premise is that the disposition of latent variables gives rise to a response in the observable ones. It is often assumed that any observed connection between manifest variables can be eliminated once the latent variables are accounted for – though this is not a valid limitation for our project. LVMs are central to our work, since we implement them to identify disease subtypes.

In an LVM, the goal is to, for each subject i, identify a vector $Z_i = \{z_{i1}, \ldots, z_{ik}\}$ of latent variables. In the case where the problem is the classification of i, we may consider $|Z_i| = 1$, so $Z_i = z_{i1}$ [Everett, 2013]. Since we're working with sequential data, e.g. a patient's status X changes over time t, an LVM typically makes use of the following notations:

- T_i: number of observations of a manifest variable for subject i.
- y_{it}: observation of a manifest variable at time t for subject i.
- x_{it}: column vector of covariates at time t for subject i.
- Z_i: vector of latent variables for subject i.

In an LVM we wish to learn the conditional distributions of the manifest vector $Y_i = \{y_{i1}, \ldots, y_{iT_i}\}$, given the covariates in $X_i = \{x_{i1}, \ldots, x_{iT_i}\}$, and a vector $Z_i = \{z_{i1}, \ldots, z_{ik}\}$ of latent variables. Concretely, the two main components of interest are:

1. $p(Y_i \mid Z_i, X_i)$ – The conditional distribution of manifest variables given X_i, Z_i.
2. $p(Z_i \mid X_i)$ – The distribution of latent variables given the covariates.

Crucially, when $T > 1$, we assume local independence of observations: manifest variables in Y_i are conditionally independent given X_i, Z_i.

2.2.1 Variational Autoencoders

A variational autoencoder (VAE) [Kingma and Welling, 2019] is a neural network, the function of which is to compress information into some latent distribution (encoding), and to reconstruct it (decoding). Though we consider the VAE to be a neural network, it is in fact comprised of two separate neural networks.

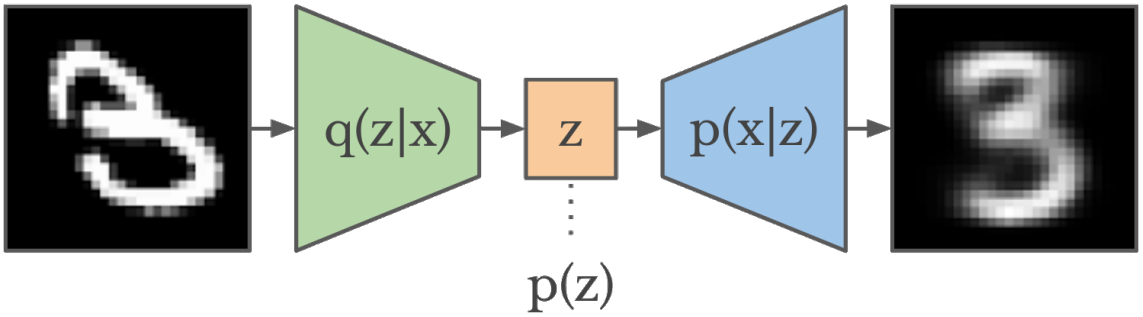


Figure 2.1: A typical VAE configuration, in this case illustrating the encoding and decoding of a digit. Image taken from [a tutorial by Danijar Hafner]

The **encoder** is the first network. It takes a datapoint x as input, and compresses it into a hidden representation z. The hidden nature of z comes from the fact that its dimensions are much smaller than those of x. The encoder is typically denoted as $q_\theta(z \mid x)$, as the space for z is stochastic. θ denotes the parameters of the encoder.

The **decoder** is the second network. Its input is the output z of the encoder. The task of the decoder is to reconstruct the datapoint x from its hidden representation z. The decoder is typically denoted as $p_\phi(x \mid z)$, where ϕ are its parameters. The goal of the decoder then, is to minimize the difference between the original datapoint x, and its reconstructed counterpart; how much information is lost in the conversion?

The total loss is $\sum_{i=1}^{N} l_i$ for N datapoints, where

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

The first term is the reconstruction loss. It gives the decoder feedback as to how well it reconstructs data. The second term is the Kullback-Leibler divergence, which describes the discrepancy between the encoder's distribution $q_\theta(z \mid x)$, and $p(z)$.

In terms of variational inference, the goal is to calculate posterior probability:

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}$$

Although intractable in this form, the posterior can be approximated with a family of distributions $q_\lambda(z \mid x)$. λ denotes which family of distributions, e.g if q were Gaussian it would mean λ is the mean and variance of the latent variable of each datapoint: $\lambda_{x_i} = (\mu_{x_i}, \sigma^2_{x_i})$

How well the variational posterior $q(z \mid x)$ approximates the true posterior $p(z \mid x)$ can again be measured with Kullback-Leibler divergence:

$$\mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x))$$

Because of the intractability of this function, VAE implementations find some equivalent function to optimize for – one that is suitable given the context of the VAE.

There are many examples where VAEs have been used and showed promising results in different domains of latent variable modeling [Rissanen and Marttinen, 2021, Louizos et al., 2017, Wang and Cunningham, 2020, Sachdeva et al., 2019]. The latent representation produced by the encoder naturally translates to the patient subtype Z in our work. We can also utilize the reconstruction of Z performed by the decoder to produce a prediction of treatment outcomes. The flexibility of the neural network architecture also allows for the combination of different strategies when using VAEs, such as combining them with recurrent neural networks, GMMs, or to generally condition them in some ways.

2.3 Sequence-to-Sequence models

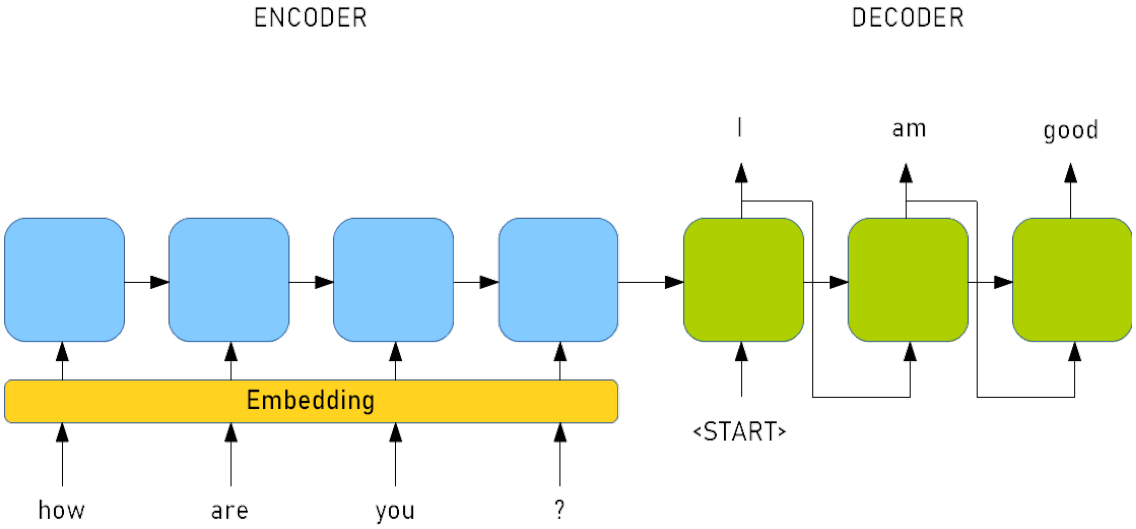


Figure 2.2: A basic seq2seq model, here applied in a chatbot. The embedding layer converts each word in the sentence into a vector of fixed size.

On its own, a VAE does not handle sequential data. However, it is possible to make it compatible using sequence-to-sequence models.

Sequence-to-sequence models [Sutskever et al., 2014] – commonly called *seq2seq* – are a group of machine-learning methods commonly used for natural language processing. Like VAEs, they feature an encoder-decoder structure, the difference being that the encoder and decoder consist of several steps, while a VAE typically only has one step for each. The intermediate steps in seq2seq have inputs leading from the outputs of previous ones. The output of one intermediate step therefore depends on previous ones.

With the use of recurrent methods such as *long-short-term memory* (LSTM) or *Gated Recurrent Units* (GRUs), seq2seq turns one sequence into another. The use of LSTM or GRUs is to solve the *vanishing gradient problem* – a phenomenon where

gradients of early elements in a sequence are lost, giving disproportional influence to the last elements of that sequence. Machine translation is perhaps the most intuitive application of seq2seq, but it is also commonly used in chat bots, text summarization, and image captioning.

Since medical records are sequential in nature, seq2seq models are also applicable to our work. Using a VAE, rather than encoding and reconstructing single patient states and treatment outcomes, we wish to do so for entire sequences of treatment. By thus employing a seq2seq model with a VAE in the middle, we make this possible.

2.4 ADCB

The Alzheimer's Disease Causal estimation Benchmark [Kinyanjui and Johansson, 2021] or *ADCB* fits a longitudinal causal model of patient variables to real data. As such it combines the strength of data-driven simulators with those of hand-crafted components. It features tunable parameters for changing the properties of the system, affecting the difficulty of the benchmark. We use ADCB as a simulator of synthetic patient data – data which is used to train the neural networks of our solution.

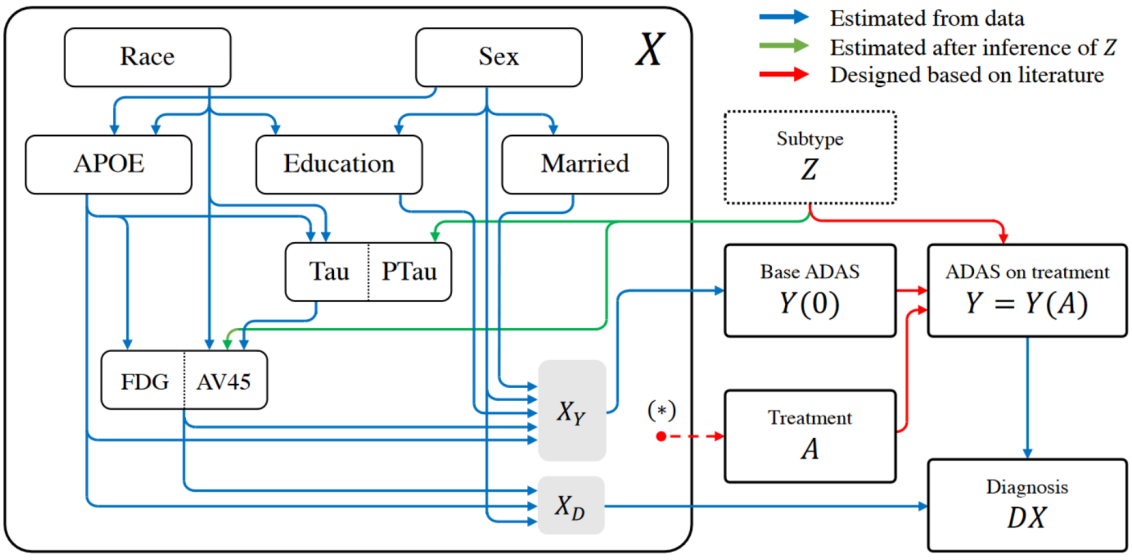


Figure 2.3: Assumed causal structure of ADCB at a single time point at baseline. Each arrow indicates a causal dependency; the color represents how the mechanism was determined.

Each generated data point features a fixed number of columns. In figure 2.3, these columns can be observed: each white box with a solid black border is such a column; the arrows between boxes indicate how features depend on eachother. The columns include universal features such as race and sex, but also medical ones such as diagnosis DX and treatment effect Y, among others. Some features are less obvious: $ADAS$ is a score between 0-83, where higher scores indicate worse cognitive function; $PTau$ and Tau are levels found in the cerebrospinal fluid that indicate

neuronal degeneration; FDG is a measurement of cell metabolism, where cells affected by Alzheimer show reduced metabolism; $AV45$ is one measurement of amyloid plaques in the brain. Further information regarding how data is fit can be found in Kinyanjui and Johansson [2021].

3

Method

The goal of our project is to develop an efficient method for finding optimal treatment for patients. We wish to accomplish this in two steps:

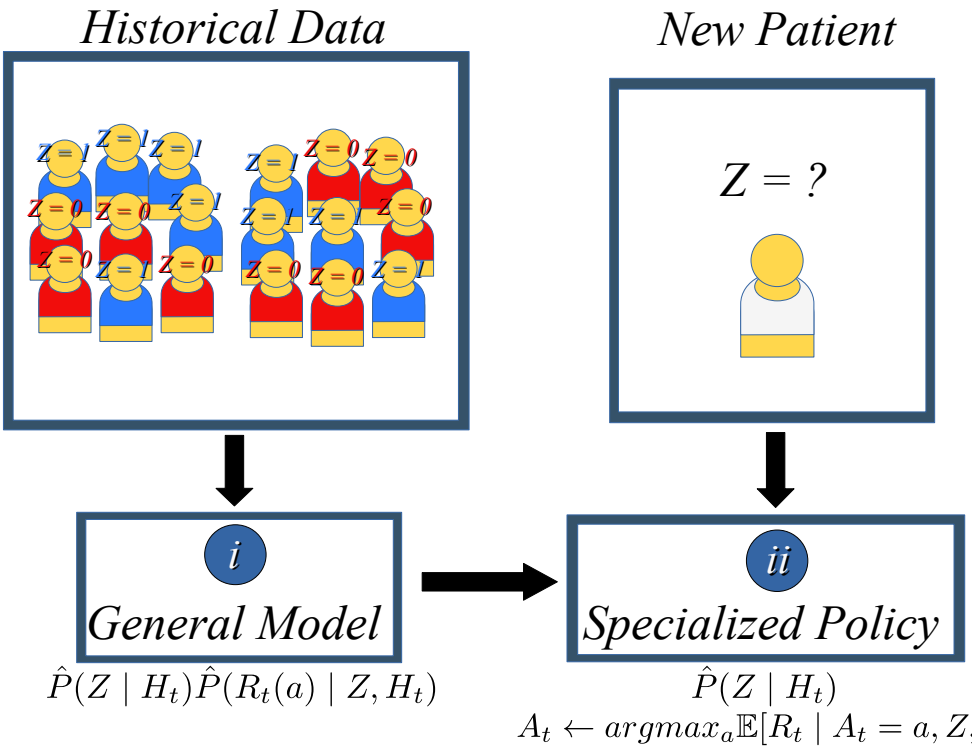


Figure 3.1: In the historical data we have access to patients who have received multiple rounds of treatment. We train a model offline on this data, allowing the model to learn about how different treatments affect patients without having to actually subject a new patient to potentially poor treatments. The model utilizes these observations to group the patients into different subtypes (0/1 or Red/Blue in the Figure). Now whenever a new patient arrives, the model can use its understanding gained from previously observed patients to efficiently categorize a new patient, without needing as many rounds of treatment before an optimal treatment is found.

i) Train a sequential model off-policy, on historical data. This will help us find a generalization of different patients – how do patients respond to a particular

treatment, on average? We believe that we may be able to identify the average effect of treatments should we be able to identify the patient subtype (Z). With the trained model in place we can use the model's prediction of any new patient's subtype to suggest an initial treatment method.

ii) For each patient, utilize the trained model in a latent bandit to recommend treatments. Using the information from the general model along with the new information gained from performing treatments, we learn a specialized policy that suggests treatments which are tailored to the conditions of individual patients, we also learn this new policy faster than if we had started with a blank slate for each new patient.

We assume that each patient has a latent type Z. Within a set of patients, there may exist subpopulations between which disease prognoses differ considerably; Z describes to which of these subpopulations the patient belongs. We therefore assume there is a causal relation between Z and treatment outcome, as well as between Z and the medical attributes of the patient. In **i)** we aim to identify which Z exist in a population of patients, and how they affect treatment outcome. This yields a general model for each Z. In **ii)** we wish to – for each patient – use this knowledge to recommend further treatments, while simultaneously obtaining a better understanding of Z.

To describe our work, we use the following notation: A patient has a particular feature tensor of medical attributes X. This tensor may contain data regarding the patient's general status, such as age and education, but naturally also the current symptoms of the patient. Much of the data in X, symptoms included, change over time – usually when treatment is performed. Therefore, we instead denote the attributes as X_t – the medical attributes of the patient at time step t. We refer to X_t as the *context*. X_t helps us in identifying Z, but it also partially helps us with identifying the optimal treatment A^*. This means that with only X_t we still require multiple rounds of treatments in order to identify the optimal treatment. In other words, X_t does not identify the optimal treatment alone, but rather help us with identification of Z.

In a latent bandit setting we think of medical treatments as actions A. An action A_t corresponds to the medical treatment performed at time step t. $\mathbb{A}$ is the set of different treatments available {1,2,...,K}, such that for any action A performed, $A \in \mathbb{A}$. When a treatment is performed it yields some sort of change – good or bad – in the status of the patient. In a latent bandit we refer to this treatment outcome as the reward R_t from taking an action A_t.

At time step t for each patient, there is for every $i < t$ a tuple (X_i, A_i, R_i) denoting the context, action, and reward at i. The tuples combine to build a history H_t:

$$H_t = \{(X_1, A_1, R_1), \ldots, (X_{t-1}, A_{t-1}, R_{t-1}), X_t\}$$

As is typical of a bandit model, we wish to maximize the cumulative reward (3.1),

or equivalently, minimize the cumulative regret (3.2) of performing actions.

$$\textbf{maximize}\, \mathbb{E}[\sum_{t=1}^{T} R_t] \tag{3.1}$$

$$\textbf{minimize}\, \mathbb{E}[\sum_{t=1}^{T} R^*] - \mathbb{E}[\sum_{t=1}^{T} R_t] \tag{3.2}$$

R^* denotes the reward from performing the action which yields the highest expected reward. Regret is simply the measurement of how far away an action is from the optimal one. In that sense, should we choose the optimal action, the regret would be 0. In a regular contextual bandit setting, the bandit is trained to learn the probability $P(R \mid X, A)$, where X denotes a context tensor which may or may not be dependent on time. In the time dependent case, this would instead be denoted $P(R_t \mid X_t, A_t)$. Assuming that each previously observed action, context, and reward needs to be accounted for when deciding on an action, what we are after is the distribution $\mathrm{P}(R_t \mid H_t)$.

Estimating this distribution can become intractable if the history becomes too long to efficiently summarize. In our approach, we condense the history into a fixed unobserved variable Z, which we believe impacts the context and reward for each time step. If we can identify $P(Z \mid H_t)$ we can sample Z' from that distribution and instead only compute the probability $P(R_t \mid X_t, A_t, Z')$, we would find ourselves in a setting identical to the latent bandit. However, as $P(Z \mid H_t)$ still relies on the entire history it might seem we would experience the same computational difficulties as in the regular case, we avoid this by utilizing the encoder from a VAE to provide us with $Z' \sim P(Z \mid H_t)$, and the decoder of this VAE to obtain $\mathbb{E}[R_t \mid X_t, A_t, Z')]$

At our disposal, we have synthetic data generated from a simulator, which itself contains representations of the latent types we wish to model with an LVM, or rather use in a comparison with our results. We find that Alzheimer's disease is suitable for study due to the fact that it is a long-lasting illness where many steps of treatment are performed.

3.0.1 Data

The data is generated from the aforementioned simulator *ADCB* which is causal estimation benchmark for Alzheimer disease. With data from this simulator we can restrict the model to the simplest of circumstances, while utilizing the simulator's clearly defined features. Most importantly, the simulator gives us access to outcome-data for any given treatment, alongside any given context. This is something which we cannot obtain from real-world data. How we use the data is further explained in 3.2.3.

3.1 Evaluation

As previously mentioned, we will be using simulated data for evaluation of our model. The main reason for this is the problem that would arise should we have utilized real-world data directly. If we were to use real-world data the bandit may suggest a treatment which is not present in the data, in that case we would have no way to evaluate the quality of the recommendation. Evaluating the model is not as simple as just measuring the accuracy, since there may be many different aspects of the model to consider. For our evaluation, we choose to focus on three different aspects:

1. **How close is the bandit's policy to an optimal policy? / Does the bandit learn an optimal policy?**
2. **How well does the LVM encapsulate patient types?**
3. **Do we achieve faster personalization?**

To answer **1.** we will compare the treatment-suggestions of our model to the treatment-suggestions from other bandit algorithms. Specifically we will compare the model with a regular Multi-Armed Bandit, a Linear Contextual Bandit, and a bandit with a perfect model.

To estimate **2.** we will compare the different values of Z to the different latent variables present in the simulator. Are the patients grouped in a similar manner to those in the simulator?

We can measure **3.** by the number of actions the bandit has to take before converging to some policy.

3.2 Process

This section describes the suggested model: the algorithm we intend to use for our bandit, as well as our latent variable model. We describe these as two stages:

1. VAE stage - An LVM is fit to historical data. This is done using a sequential adaptation of VAEs.
2. Bandit stage - For each patient, a latent bandit model is learned using the LVM as basis. Using a modified version of the mTS algorithm, this assumes a causal structure between variables. An overview of this structure can be observed in Figure 3.2:

3.2.1 SEQ-GSM

After data preprocessing, which simply consists of computing the standard score of the features which contains continuous values, we train a model which we will henceforth refer to as a *Sequential Gumbel-softmax Variational Autoencoder* (*SEQ-*

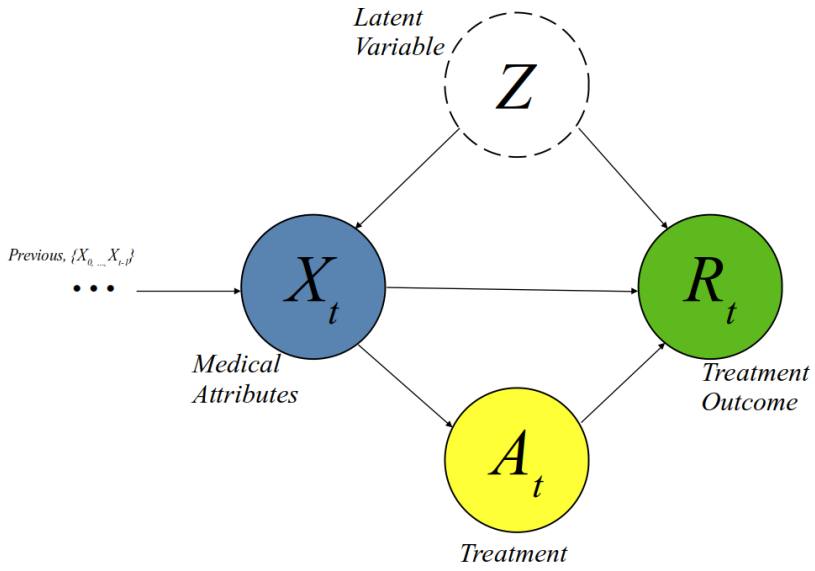


Figure 3.2: We assume a causal structure as illustrated above. Arrows indicate a causal relationship between variables. The dashed line around Z is to denote that it is a latent variable.

GSM). The model is a neural network which combines the logic inside a typical VAE, with that of the recurrent structure found in seq2seq models. The architecture of the model can be seen in Figure 3.3.

A typical VAE would feature a single encoder-decoder which data is processed through; this is useful in domains such as image generation, but insufficient for sequential data such as treatment effects over time. In this project we are working with sequential data in the form of medical treatments and outcomes, and the goal of the model is to efficiently learn to estimate the latent variable Z, which is accomplished by the following encoding-decoding process:

Encoding

The goal of the encoding process is to compress the information from every tuple (X_i, A_i, R_i) in a sequence $H_t = \{(X_0, A_0, R_0), \ldots, (X_{t-1}, A_{t-1}, R_{t-1}), X_t\}$ which will then be used to estimate the value of the latent variable. Each $X_i \in H_t$ has F features. The architecture of the encoder consists of a fully connected layer $X_i \rightarrow$ FC(300), a ReLu activation function, and a GRU, (300, 300) $\rightarrow$ (300, 300). The GRU itself creates two outputs, one which we will disregard in each time step, and one which will be used as input for the next encoding layer. The last act of the encoder is to transform the output into a tensor of size $B \times K$, where B is the batch size and K is the upper limit of values that Z can assume. Z is used as input for the decoder.

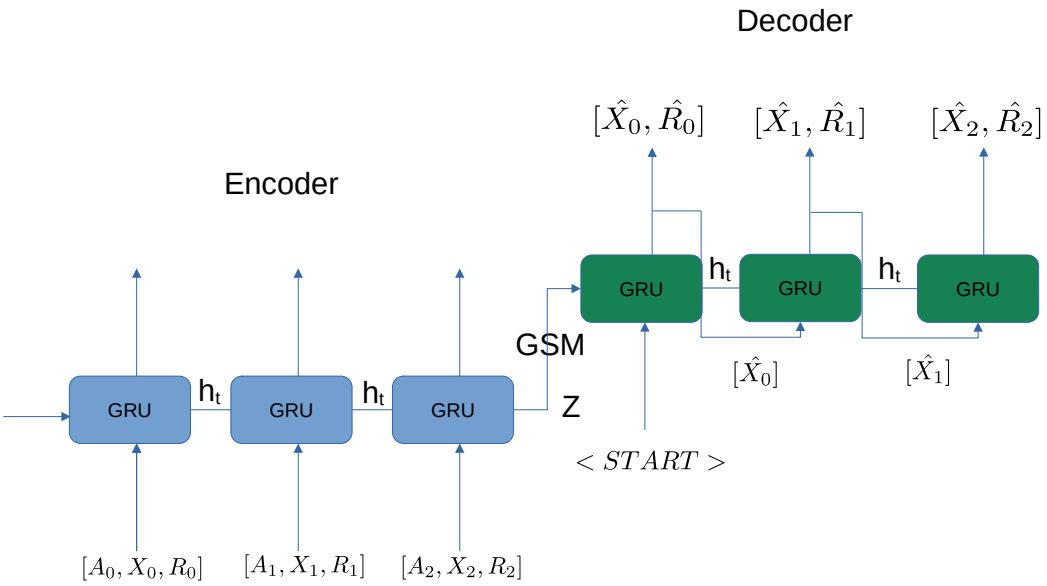


Figure 3.3: Architecture of the neural network used in the SEQ-GSM VAE.

Gumbel-softmax

Using the features currently available, the encoder produces an estimate of Z, the certainty of which may be lesser or greater. We consider this estimate to be some categorical distribution $\mathbb{P}$, from which we sample the discrete variable Z. This introduces a problem in the context of neural networks, since backpropagation is not possible for stochastic or discrete processes. If we consider $\mathbb{P}$ to be continuous, we can approximate its samples to a discrete space post hoc. The distribution is stochastic, but with the use of *Gumbel-softmax* introduced in Jang et al. [2016], we can reparameterize $\mathbb{P}$ so that it becomes differentiable.

Figure 3.4 features a visual representation of how Gumbel-softmax works. We can observe the log probabilities $\log \alpha_i$ of $\mathbb{P}$ in the bottom left corner. These are deterministic, and compatible with backpropagation. The log probabilities are combined with noise G_i; these are sampled from the Gumbel distribution – described in Gumbel [1941] – and are thus stochastic. This linear combination is run through the softmax function to retrieve the class with the greatest probability. We can use the added factor λ to control how closely the Gumbel-softmax distribution resembles $\mathbb{P}$.

Decoding

The goal of the decoding process is to use the information about the latent variable to try and reconstruct the sequence H_t – The motivation behind this being that the

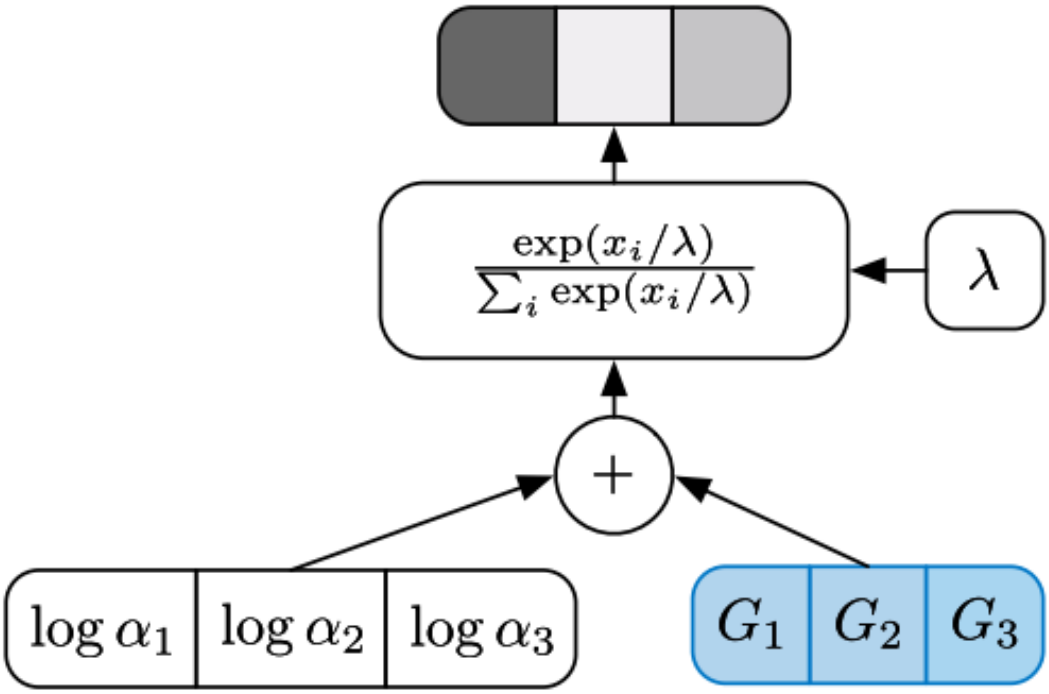


Figure 3.4: The reparameterization trick recasts the distribution into a linear combination of probabilities. Image taken from Jang et al. [2016]

more accurately we can reconstruct treatment outcomes, the easier it becomes to recommend the treatment that corresponds to the best outcome. The architecture of the decoder consists of a combination of eleven linear layers, which use-cases are described more precisely below, and a GRU. Unlike in the encoder, in the decoder it is the first part that stands out, since it is here the $B \times K$ tensor is passed through a linear layer to create the hidden state, a $B \times 300$ tensor h_t. A concatenated tensor $[h_t, \hat{X}_t]$ is passed through several parallel linear layers, where each linear layer is intended to represent a different treatment $A \in \mathbb{A}$. These linear layers each produce a $B \times 1$ tensor, corresponding to some treatment outcome $\hat{R}_t(A)$.

To explain why we do not pass the treatment indicator (A) as part of input to the decoder we refer to the paper Shalit et al. [2017]. In the proposed model from the paper (TARNet), the treatment indicator is not passed as a feature either, but rather parallel linear layers are used to represent the different possible treatments. The motivation behind this is that if the treatment indicator is passed as a feature alongside all other potentially high-dimensional features, it's impact on the outcome may be minimal, or lost entirely. To circumvent this, we use separate layers to perform distinct predictions for each treatment, ensuring that the effect of different treatments is not lost among the other features of the data.

Finally, the decoder at each step returns a triplet consisting of $\hat{R}_t$ which is a tensor of each predicted outcome $\hat{R}_t(A)_{A \in \mathbb{A}}$, $\hat{X}_t$, and h_t. The next decoder step uses h_t as its hidden state and a tensor $[\hat{X}_t]$ as input.

Loss

We measure the loss of our reconstructions by calculating the mean-square error (MSE). However, as mentioned above we reconstruct the entire outcome tensor $\hat{R}_t$ for each possible action, but since only one action is observed at a time we calculate the loss by onehotting this tensor such that our outcome-*loss tensor* can be defined as $L_y = \left[(R_t - \hat{R}_t)^2\right] \cdot [0 \text{ if } A \neq a \text{ else } 1 \text{ for } A \in \mathbb{A}]$ where a denotes the observed action. The loss is calculated similarly for each feature $f \in X_t$ such that $L_x = \sum \left[(f - \hat{f})^2\right]$. The final loss is then simply: $L = L_x + L_y$

3.2.2 Multi-Armed Bandit with Thompson Sampling

Once the SEQ-GSM has been trained, we move on to the bandit stage where we employ mTS [Hong et al., 2020]. This is a loop of using Thompson sampling to obtain Z from a prior distribution, and taking the best action available given this sample of Z. Lastly, from the treatment outcome, a posterior distribution is calculated – this posterior will serve as the prior in the next iteration.

Although we apply mTS, we deviate somewhat from its standard setup in our assumption of relationships between variables. This can be observed in Figure 3.5. More precisely we assume that the X_{t-1} impacts X_t, and also that Z impacts X_t at every timestep t = {0,1,2,...,T}.

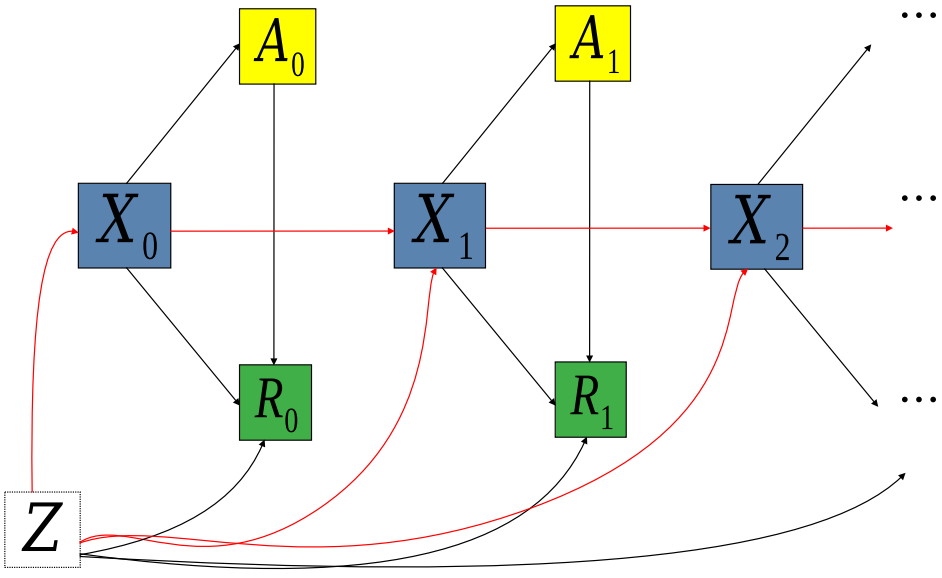


Figure 3.5: Expanding upon the structure described in Figure 3.2, we can describe the relationships necessary to perform mTS. The red arrows describe relationships that are not present in typical mTS.

At each iteration of mTS, we rely on a prior distribution $P(Z)$ of the latent variable. Starting off, we know nothing of the patient; at this point $P(Z)$ is simply the

generalized model obtained from the VAE stage. As the iterations go on the bandit will specialize, and this distribution will be updated to better reflect our certainty of Z. We wish to find a posterior distribution $P(Z \mid H_t)$ conditioned on a patient's history $H_t = \{(X_0, A_0, R_0), \ldots, (X_{t-1}, A_{t-1}, R_{t-1}), X_t\}$. Here, there is a decision to be made regarding how this should work:

i) Like in mTS, we could use the decoder to model the entire history; using Bayes' rule:

$$P(Z \mid H_t) = \frac{P(H_t \mid Z)P(Z)}{P(H_t)}$$

Since $P(Z)$ is given, and $P(H_t)$ can be eliminated on account of normalization, we can find the posterior $P(Z \mid H_t)$ by computing the likelihood $P(H_t \mid Z)$. Using the expanded causal structure from figure 3.5, we can model the entire history:

$$P(H_t \mid Z) \propto P(Z) \left[\prod_{\ell=0}^{t-1} P(X_\ell \mid Z)P(R_\ell \mid X_\ell, A_\ell, Z)\right] P(X_t \mid X_{t-1}, Z) \quad (3.3)$$

Using the decoder we could estimate each factor of this likelihood. However, the number of factors are great; there would be a degree of uncertainty in each estimation, and since each factor depends on the last, this uncertainty would be compounded into the final product.

ii) Alternatively, we could estimate $P(Z \mid H_t)$ directly using the encoder.

We have decided to use ii), since the fewer steps involved means it is less computationally intensive, and likely more accurate. Sampling from the posterior yields a new estimate $Z' \sim P(Z \mid H_t)$. With X_t and Z' as context, the decoder now estimates the best action:

$$A_t \leftarrow \underset{a}{\operatorname{argmax}}\, \mathbb{E}\left[R_t \mid A_t = a, Z', X_t\right]$$

The action taken A_t yields a reward R_t, which leads us to the next iteration of the loop with updated history $H_{t+1} = H_t \cup \{(X_t, A_t, R_t), X_{t+1}\}$. An overview can be observed in algorithm 2:

Algorithm 2 mTS with estimated posterior

Input:
Prior over latent type $P(Z)$
$H_t := \{(X_0, A_0, R_0), (X_1, A_1, R_1), (X_2, A_2, R_2), \ldots, (X_{t-1}, A_{t-1}, R_{t-1}), X_t\}$
for t $\leftarrow 1, \ldots, T$ **do**
  Sample $Z' \sim \hat{P}(Z \mid H_t)$
  Select $A_t \leftarrow \operatorname{argmax}_a \mathbb{E}(R_t \mid A_t = a, Z', X_t)$
  Update prior $P(Z) = \hat{P}(Z \mid H_t)$
end for

3.2.3 Data Pipeline

Our data pipeline (Figure 3.6) consists of generating a set amount of patients (N) with a fixed horizon (T) on which the VAE will be trained. We let N = 10000 and T = 6 for the VAE. This means that we generate 10000 patients – each receiving 6 rounds of treatment. During training of the VAE we let this sequence length be chosen randomly between 2 and 6. This decision was made as to not let the model be trained on sequences of just a single length, as that might lead to poor performance in earlier stages of treatment suggestions. These 10000 patients are split intro three separate datasets for training, validation, and testing, in a 60/20/20 randomly selected split of the data. The training set is used to update the model. The loss observed in the validation set is used to determine if we need early stopping after τ epochs to prevent overfitting the model, and the loss from the test set is how we estimate performance of the model.

To test the models in a bandit setting we generate a new set of 10000 patients, split into a test set and validation set. The test set is used to compare how the VAEs with different hyper-parameters perform, from which the best performing VAE is chosen. This is done by looking at the average regret achieved by the VAEs over a fixed amount of bandit runs. After the best performing VAE from the test set is chosen we use the validation set to compare this VAE with other algorithms which is then presented as results.

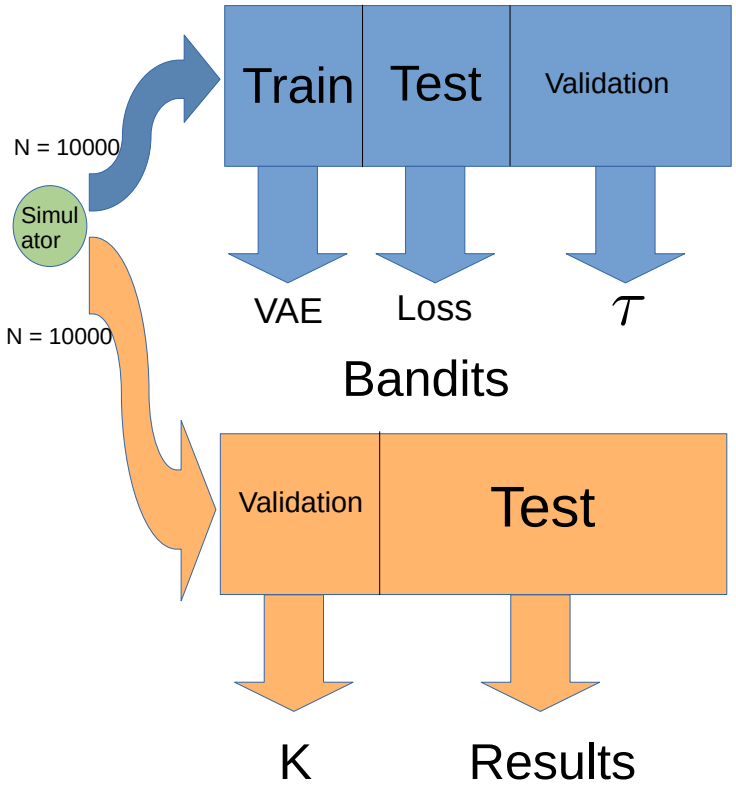


Figure 3.6: Data generation pipeline - Two separate datasets with 10000 patients each are created. One to be used for the VAE, and one to be used for the bandit. The VAE data is used to determine early stopping, measure reconstruction loss, and to update the weights in the network. The bandit dataset is used for determining a suitable K and to compare the model with other bandit algorithms.

4

Results

In the following two chapters we address the questions presented in section 3.1. Here in chapter 4 we will present results necessary to answer the questions, and interpret their meaning. In chapter 5 we will talk further about these questions, and discuss our project in a broader context.

4.1 Reconstruction

Since the core of our model is composed of a VAE – the function of which is to reconstruct data from an encoding – the first part of evaluation revolved around how well we could reconstruct the data from the simulator. As we know, the simulator data consists of sequential data of a specified length for each patient. One data point in a sequence can be seen as a vector $[A_t, X_t, R_t]$. The encoder produces a prediction $\hat{Z}$ which it passes on to the decoder, and the goal of the decoder is to reconstruct $[X_t, R_t]$ such that the output of the decoder $[\hat{X}_t, \hat{R}_t] \approx [X_t, R_t]$.

4.1.1 Comparing models in terms of loss

Two of the hyper-parameters of the model are the most significant in terms of comparing models between one another. The first comparison is made between models using a continuous latent space, and models using a discrete latent space. The second parameter is the size of the latent space. Instead of exhaustive testing we investigated models with sizes K = [2, 10, 100, 200, 500, 1000]. Comparing the models in Figure 4.1 with the models in Figure 4.2 we see that for both continuous and discrete $K = 2$ the models perform about equally in terms of loss; however, for any K larger than that the continuous models outperform the discrete models. The large gap between training loss and testing/validation loss is nothing unexpected. Given that we are in a regression setting the training loss can become very small if the model starts to learn the noise that is present in the data. As we will not learn the noise for the test- and validation-set, the validation- and test-loss will be higher. Furthermore, there may be features in the context which are not realistic to accurately predict as there may not exist a causal relation between them. Such features will also add to the gap between the different losses.

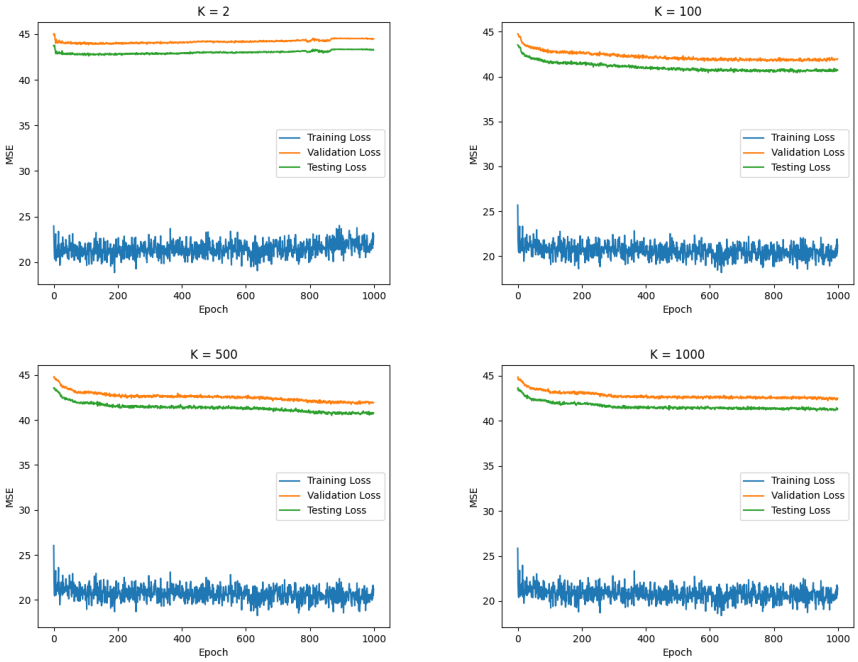


Figure 4.1: For the discrete models present in this figure we can see that the training and validation loss remain relatively unchanged.

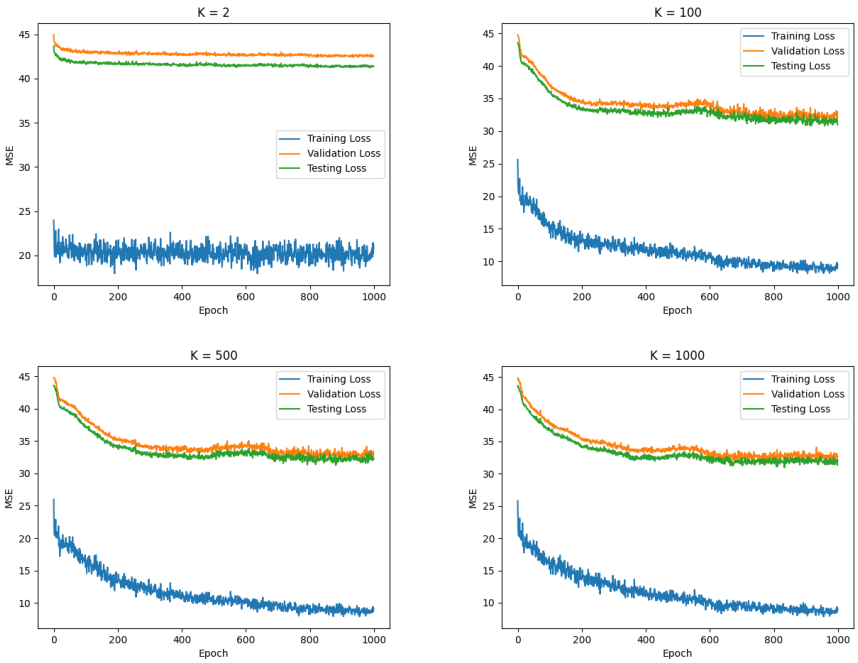


Figure 4.2: For the continuous models present in this figure we can see that the training and validation loss change steadily for more epochs than in the discrete case, and that they also reach a lower minimum of training loss as well as validation loss.

4.2 Bandit Performance

When evaluating bandit performance we first look at whether or not the bandit converges to some policy, and second if that policy is equivalent to or similar to an optimal policy.

4.2.1 Comparing models in terms of regret

Given that the goal of the project is not to minimize the loss acquired by the models, but rather to find a model which performs as well as possible in a bandit setting, we further compared the average regret when running the bandit using the different trained VAEs as models.

The discrete models achieved very similar average regret for every model except the one that was trained with K = 2 which can be seen in Figure 4.3. The same comparison for the continuous models showed more promising results as evident in Figure 4.4. Here we see that K = 1000 and K = 500 significantly outperform the other models.

We believe this change in performance arises from the fact that for a larger latent space, the model can capture more features present in the context, rather than just trying to estimate Z. This means that the $\hat{Z}$ used as decoder input, likely contains more information about the individual than the discrete Z in the simulator could do on it's own. Given that there still exists a lot of variance between the features present in the context, this sort of dimension increase could possibly capture such variance, therefore making the model more personalized. With this in mind, we choose to focus on of continuous models with a large latent space for policy comparison.

4.2.2 Policy

For policy evaluation we present five different algorithms. A Multi-armed Bandit (MAB) [Lattimore and Szepesvári, 2020] with Thompson Sampling (TS) [Agrawal and Goyal, 2012], a Linear Contextual Bandit (LCB) [Lattimore and Szepesvári, 2020] with TS [Agrawal and Goyal, 2012], a perfect model mTS, and two versions of our proposed learned model mTS. The first version of the learned model mTS algorithm is the estimated posterior mTS-algorithm (Algorithm 2). We also introduce a second version of the algorithm, which is identical in the encoding process but differs in the decoding process. In the first version of Algorithm 2 we only utilize the context for the current timestep and perform one step of decoding to perform prediction, whereas in the second version we run the decoder on all of the previous witnessed data and use the predictions from the last timestep. What we can observe in Figure 4.5 is that the first version of the algorithm strongly outperforms the second version, and that the second approach in fact seems to become worse the longer the history becomes. This indicates that while the encoder flourishes from a longer history to accurately reconstruct Z; the decoder performs best using only the currently observed context. We also observe that after 100 rounds of treatments the LCB has yet to come anywhere near the optimal policy, and that it is outperformed

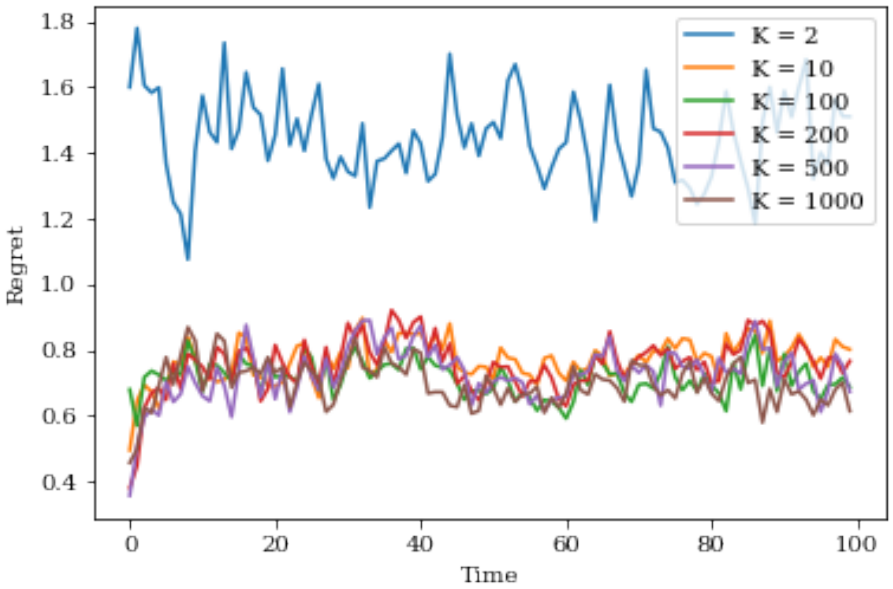


Figure 4.3: Average regret for 100 different bandit runs using VAEs trained with different discrete latent space sizes K.

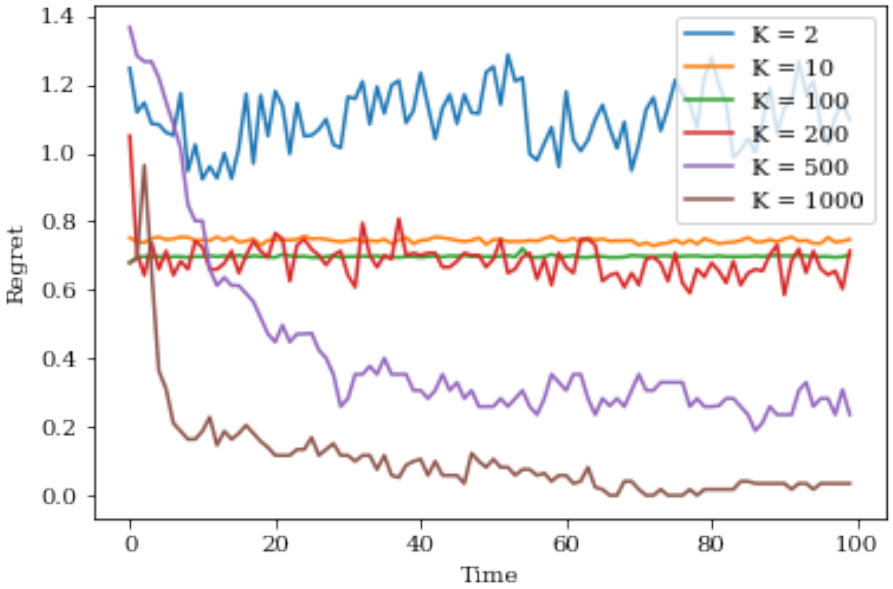


Figure 4.4: Average regret for 100 different bandit runs using VAEs trained with different continuous latent space sizes K.

by the MAB.

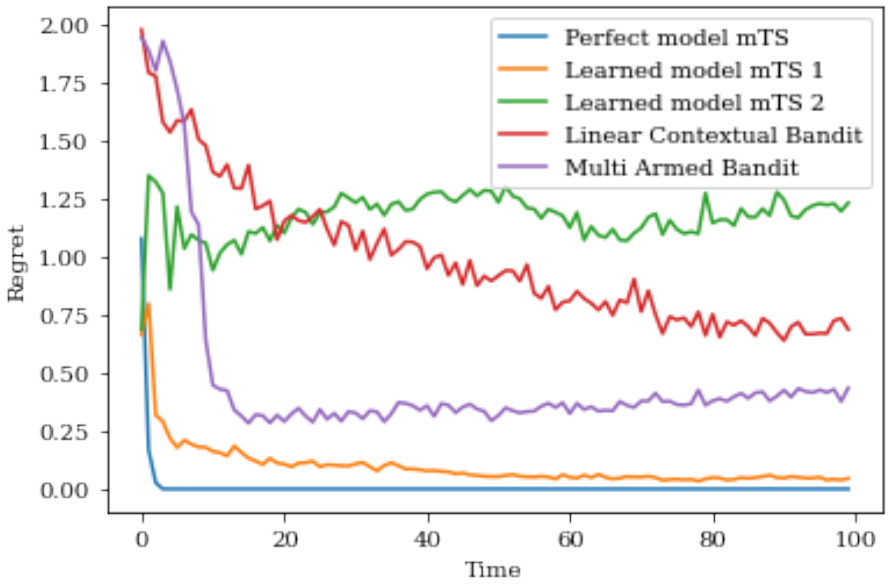


Figure 4.5: Average instantaneous regret for 500 bandit runs over 100 rounds of treatment.

We realized a possible reason for the MAB outperforming the LCB is that LCB takes into account the time-varying context X_t. Given that we know from the simulator that the context cancels out when comparing the reward between different actions the algorithm will struggle with time-varying context, increasing the time necessary to learn the optimal policy. To account for this, we investigated a version of LCB which only considers the context at timestep 0 (X_0) and observed the following results (Figure 4.6). It is difficult to determine which comparison of LCB and learned model mTS is the most fair, given that knowing that the context does not impact the reward is not granted, but rather comes from an engineering aspect of how the simulator works. In both cases however we can still see that our proposed model outperforms the LCB and MAB in early treatment rounds, and that it also converges to the optimal policy faster.

4.3 Encapsulation of patient types

As we have access to the latent types present in the simulator for every patient, we wanted to find a way to compare how the simulator groups patients, with how our model groups patients. Given that the some of the versions of the model we proposed uses a latent space of sizes up to a 1000, and also versions which utilizes a continuous latent variable, we decided to look at t-stochastic neighbor embedding (T-SNE) [Van der Maaten and Hinton, 2008] as a way to verify if the model produces reasonable clusters or not. We decided to look at the T-SNE projection for the posterior in the last timestep of treatment for 10000 patients from the simulator, using the simulator latent variable as the labels $[0, 1]$, and our posterior as data,

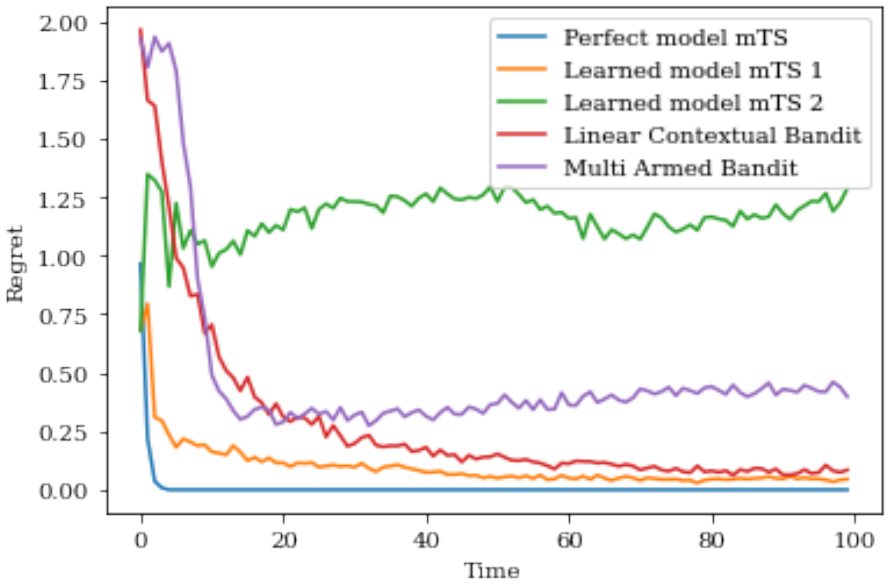


Figure 4.6: Average instantaneous regret for 500 bandit runs over 100 rounds of treatment with LCB using only context X_0

we can investigate if our considerably larger latent space can be compressed into a much smaller one. using a version of our trained model with a continuous latent space of size 500 we observed the following result in Figure 4.7. Furthermore we investigated how the posterior changes over time by investigating how the data would be clustered for T = 1 to get an idea of whether or not we actually infer something over time, or if the same clusters would be evident before the bandit has started trying treatments. We compared this with looking at clusters obtained by looking at the posterior for T = 6, which is the horizon-length for our trained models. Comparing the clusters in Figure 4.8 we can see that the data is indeed not clustered initially but rather appears somehow random, whereas by timestep 6 the data does appear to have formed a cluster similar to that for T = 100.

After observing the clusters from the T-SNE projections we can see that the model seems to divide the data into a higher number of clusters than the number present in the simulator (2). It is uncertain as to why there are more than 2 clusters but there are many reasons as to why this could happen. One possibility is that features of the data are captured which lowers the variance from context between patients, essentially dividing the Zs in the simulator into smaller subsets, capturing information about the subtype, the context, and the outcomes. This would come as a result of the fact that the encoder in the VAE, and therefore the estimated latent space, has an incentive to reconstruct both the context and the outcomes.

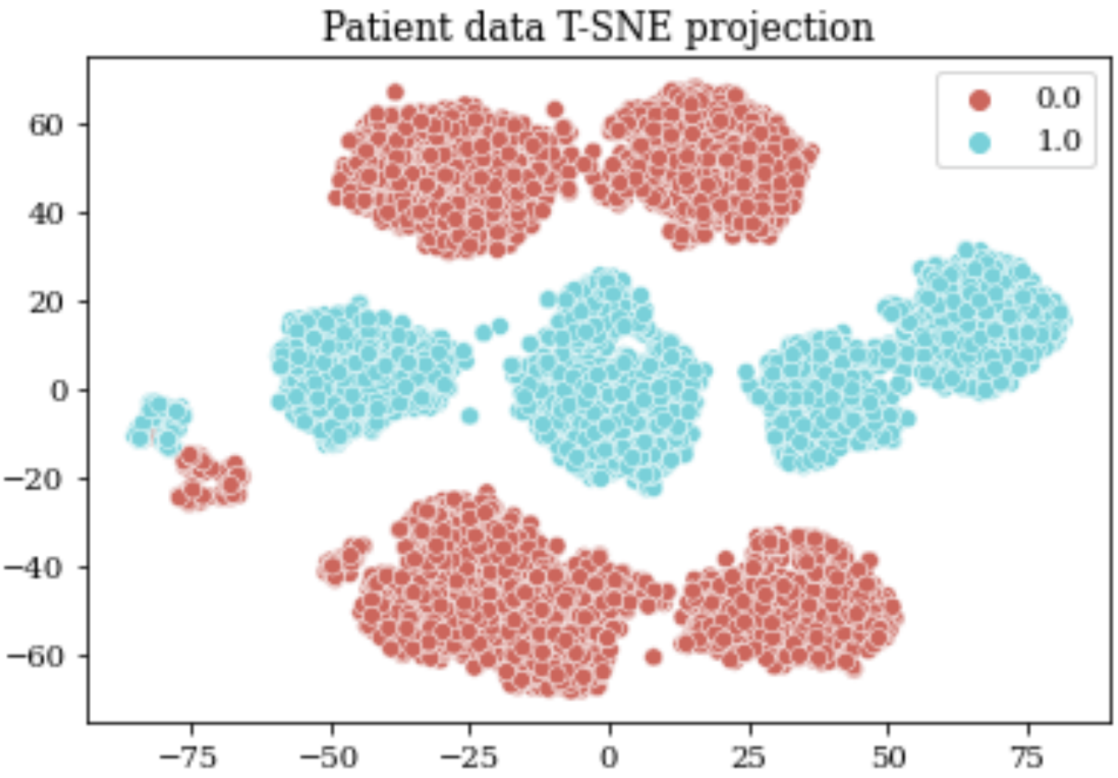
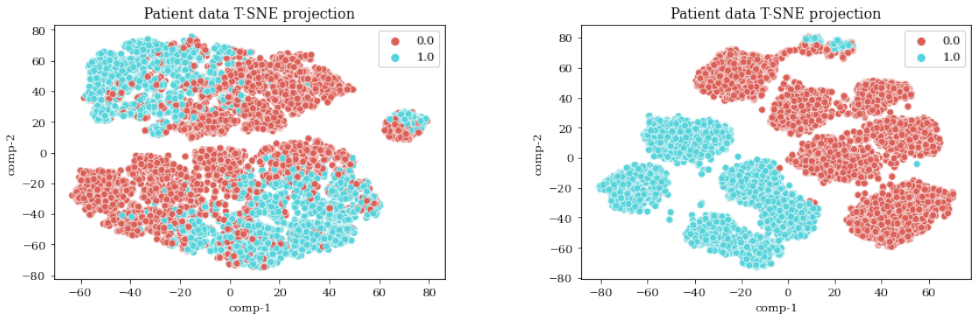


Figure 4.7: T-SNE projection for a continuous latent space of size 500, indicating clearly clustered data.



(a) Projection for T = 1 for full bandit dataset.

(b) Projection for T = 6 for full bandit dataset.

Figure 4.8: Comparison of projection of posterior for Z for different rounds of treatment.

5

Discussion

In this chapter we discuss our achieved results, elaborate on what future work we believe could be made by expanding on our research, and finally we present our conclusions.

5.1 Correlation between VAE-Loss & Bandit Performance

Comparing the L-MTS models with different latent spaces with each other our initial hypothesis was that the lower loss in the VAE, the lower the regret would be for the bandit. We can determine that the continuous models with a large latent space outperforms the discrete models both in terms of loss and in bandit performance. However, some of the continuous models which achieved a lower loss than the discrete models still performed roughly equally in the bandit setting. The most reasonable explanation for this deviation between loss and regret should be what features are most accurately reconstructed. Different latent spaces may lead to better reconstruction of separate variables. A model which very precisely reconstructs the context but is awful at reconstructing outcomes will lead to a very poor-performing model in terms of regret, but may still have very good results in terms of loss. Consequently, it is hard to determine how well a model will perform in a bandit setting by only observing the total loss measured from the offline setting.

5.2 Effects of overparameterization

Our initial presumption was that a suitable value of K would be 2. This was based on the idea that a patient having a high or low degree of amyloid plaques in the brain has a considerable effect on both their prognosis, and which treatment is considered the best [Kinyanjui and Johansson, 2021]. Despite this, the models of ours that best fit the data are the ones that have a far greater value of K, as seen in figures 4.1 and 4.2. In addition, these models specialize faster for individual patients, as seen in figures 4.3 and 4.4.

It is possible that the figures suggest our solution has managed to identify a concrete number of latent types greater than what we initially expected. It is also likely that the improved performance observed in models with greater K is due to

the phenomenon of overparameterization: a neural network that has oversized parameters often tends to perform better on unseen test data compared to one using parameters more 'normal' in size. A neural network of this type is also typically easier to optimize (even getting the training loss low that is). This usually comes with the caveat that overparameterized models take longer to identify relationships than normal models do – assuming that the normal models are able to identify these relationships in the first place. This is a point of interest for our project, since being efficient in the identification of good treatments is our goal.

Is this caveat present in our models? If it were, we would see that, at early phases of treatment, models using too great a K have a clear increase in bandit regret compared to ones using a lower K. Let us consider the first few actions taken by models – with discrete and continuous K – in figures 4.3 and 4.4, respectively.

The distinction between discrete and continuous K is an important one, since the number of possible real numbers in $[0, K]$ is far greater than the number of of integers in that interval. The discrete $K_d = 2$ is among the worse performers early. This is within the realm of plausiblility, since it is difficult for a neural network to identify relationships in such a small space. Meanwhile, the continuous $K_c = 500$ is clearly the worst early on in terms of performance. This would suggest that $K_c < 500$ are better at identifying near-optimal treatments quickly, and thus more suitable for our goals. However, this is contradicted by $K_c = 1000$. Early, it has a performance on par with smaller K, despite its size. The reason for this is unclear, but it could be that $K_c = 500$ strikes an unfortunate balance of both being too simple to identify complex relationships, and too overparameterized to perform well. At any rate, overparameterization seems to carry no downside in our project, with $K_c = 1000$ being the best both short-term and long-term.

5.3 Ethics

We have made delimitations for our solution in order to make it attainable. Although such delimitations can be a minor hurdle in other fields, in health care they are more often detrimental to the applicability of the product. For instance, we consider immediate treatment reward to be the sole metric for success, and we assume that this treatment effect remains constant in time; this, despite the fact that long-term well-being may be considered more important, and may be affected by factors relating to treatment effect. The effect of medications may change gradually with continued use, and also come with side-effects that are not so easily quantified.

Our delimitations have allowed us to achieve something of substance despite the project being quite limited in scope, but it also means that further work is required to make our solution practical; it must adhere to standards of medicine in order to ensure that the well-being of patients is not compromised.

5.4 Future Work & Conclusion

We ended our project by investigating the feasibility of what we referred to as a *Simple Decoder*. This model would be equivalent to the proposed one with the exception that the decoder disregards context and only utilizes the predicted $\hat{Z}$ as decoder-input and only reconstructs the outcomes $\hat{R}_t$. We only managed to produce a version of this model which converged to a single action at all times. Such a model should realistically be able to learn the optimal policy given that the context does not impact the reward, therefore future work could include further investigating such an approach.

Another area of future work would be to investigate utilizing the the approach from this project, which is a model with a large continuous latent space, to further train a model with a small discrete latent space. This practice is called distilling a neural network [Hinton et al., 2015]. The purpose of the method is that a smaller latent space should reduce the complexity of the model, which might have a positive impact on performance (execution time).

Other areas of future work could include looking at a different disease than Alzheimer, given that the model itself has no constraints for focusing on this disease in particular. Such diseases would of course preferably be those which also require several steps of treatment, one such disease which immediately comes to mind is Cancer. There are of course many other diseases requiring multiple rounds of treatment for which our proposed model could potentially be applicable. Which those diseases might be remains to be investigated.

Furthermore, more research definitely needs to be done on real data, rather than synthetic. Nevertheless, given that one can not find observations for different treatments to the same extent in real data nor experiment on real patients, such work becomes increasingly difficult.

One could also experiment with imputing data, given that we generate the data we always have *perfect information* about patients, in the real world that is not typically the case. One could simply try imputing data points at random to make sure that the model still learns an optimal policy. Nevertheless, this type of research would become most interesting in the bandit setting, with the objective of investigating if we can still perform good treatments with missing data.

We believe that we have demonstrated results which answer all of our research questions. Firstly, we can conclude that the proposed model does in fact learn a policy which is optimal, or at least very close to optimal. Secondly, we can observe distinguishable clusters when using T-SNE, which we believe shows that the LVM does perform patient encapsulation. Thirdly, as we could clearly observe a faster convergence time for our proposed algorithm than the other bandit algorithms, we see that faster personalization has been achieved. Finally, we believe that we have shown that there is a benefit of utilizing a learned model in the domain of finding optimal treatments efficiently, which is what we set out to investigate.

Bibliography

Revlin Abbi, Elia El-Darzi, Christos Vasilakis, and Peter Millard. A gaussian mixture model approach to grouping patients according to their hospital length of stay. In *2008 21st IEEE International Symposium on Computer-Based Medical Systems*, pages 524–529. IEEE, 2008.

Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*, pages 39–1. JMLR Workshop and Conference Proceedings, 2012.

Dennis A Benson, Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers. Genbank. *Nucleic acids research*, 41 (D1):D36–D42, 2012.

Bibhas Chakraborty and EE Moodie. Statistical methods for dynamic treatment regimes. *Springer-Verlag. doi*, 10:978–1, 2013.

Philip L De Jager. Identifying patient subtypes in multiple sclerosis and tailoring immunotherapy: challenges for the future. *Therapeutic Advances in Neurological Disorders*, 2(6):369–377, 2009.

B Everett. *An introduction to latent variable models*. Springer Science & Business Media, 2013.

David Fenstermacher, Craig Street, Tara McSherry, Vishal Nayak, Casey Overby, and Michael Feldman. The cancer biomedical informatics grid (cabig tm). In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pages 743–746. IEEE, 2006.

John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011.

Arthur Guez, Robert D Vincent, Massimo Avoli, and Joelle Pineau. Adaptive treatment of epilepsy via batch-mode reinforcement learning. In *AAAI*, pages 1671–1678, 2008.

Emil Julius Gumbel. The return period of flood flows. *The annals of mathematical statistics*, 12(2):163–190, 1941.

Samuel Håkansson, Viktor Lindblom, Omer Gottesman, and Fredrik D Johansson. Learning to search efficiently for causally near-optimal treatments. *arXiv preprint arXiv:2007.00973*, 2020.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Joey Hong, Branislav Kveton, Manzil Zaheer, Yinlam Chow, Amr Ahmed, and Craig Boutilier. Latent bandits revisited. *arXiv preprint arXiv:2006.08714*, 2020.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2016. URL `https://arxiv.org/abs/1611.01144`.

Michael N Katehakis and Arthur F Veinott Jr. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

Newton Mwai Kinyanjui and Fredrik D Johansson. Adcb: An alzheimer's disease benchmark for evaluating observational estimators of causal effects. *arXiv preprint arXiv:2111.06811*, 2021.

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

Christos Louizos, Uri Shalit, Joris Mooij, David Sontag, Richard Zemel, and Max Welling. Causal effect inference with deep latent-variable models. *arXiv preprint arXiv:1705.08821*, 2017.

Catherine R Planey and Olivier Gevaert. Coincide: A framework for discovery of patient subtypes across multiple datasets. *Genome medicine*, 8(1):1–17, 2016.

Aniruddh Raghu, Matthieu Komorowski, Imran Ahmed, Leo Celi, Peter Szolovits, and Marzyeh Ghassemi. Deep reinforcement learning for sepsis treatment. *arXiv preprint arXiv:1711.09602*, 2017.

Severi Rissanen and Pekka Marttinen. A critical look at the consistency of causal estimation with deep latent variable models. *Advances in Neural Information Processing Systems*, 34, 2021.

Noveen Sachdeva, Giuseppe Manco, Ettore Ritacco, and Vikram Pudi. Sequential variational autoencoders for collaborative filtering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 600–608, 2019.

Rob Saunders, Joshua EJ Buckman, and Stephen Pilling. Latent variable mixture modelling and individual treatment prediction. *Behaviour research and therapy*, 124:103505, 2020.

Andrew J Schaefer, Matthew D Bailey, Steven M Shechter, and Mark S Roberts. Modeling medical treatment using markov decision processes. In *Operations research and health care*, pages 593–612. Springer, 2005.

Uri Shalit, Fredrik D Johansson, and David Sontag. Estimating individual treatment effect: generalization bounds and algorithms. In *International Conference on Machine Learning*, pages 3076–3085. PMLR, 2017.

Aleksandrs Slivkins. Introduction to multi-armed bandits. *arXiv preprint arXiv:1904.07272*, 2019.

D.H. Solomon, C. Xu, J. Collins, S.C. Kim, Elena Losina, Vincent Yau, and Fredrik D. Johansson. The sequence of disease-modifying anti-rheumatic drugs: pathways to and predictors of tocilizumab monotherapy. 2021.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Guy Tennenholtz, Uri Shalit, Shie Mannor, and Yonathan Efroni. Bandits with partially observable confounded data. In *Uncertainty in Artificial Intelligence*, pages 430–439. PMLR, 2021.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

Huan-Hsin Tseng, Yi Luo, Sunan Cui, Jen-Tzung Chien, Randall K Ten Haken, and Issam El Naqa. Deep reinforcement learning for automated radiation adaptation in lung cancer. *Medical physics*, 44(12):6690–6705, 2017.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Yixin Wang and John Patrick Cunningham. Posterior collapse and latent variable non-identifiability. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2020.

Yufan Zhao, Donglin Zeng, Mark A Socinski, and Michael R Kosorok. Reinforcement learning strategies for clinical trials in nonsmall cell lung cancer. *Biometrics*, 67 (4):1422–1433, 2011.