TECHNICAL UNIVERSITY OF CRETE

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DIPLOMA THESIS

Adversarial Learning in Statistical Dialogue Systems

Author:
Dialektakis George

Commitee:
Associate Professor Michael G. Lagoudakis
Associate Professor Georgios Chalkiadakis
Dr. Vassilios Diakoloukas

A thesis submitted in partial fulfillment of the requirements for the
Diploma degree in Electrical and Computer Engineering
July 2020

Adversarial Learning in Statistical Dialogue Systems

by George Dialektakis

Abstract

In the past few years, the machine learning community has shifted its attention in Generative Adversarial Networks (GANs) and has shown their enormous potential in image, video and audio generation. Nevertheless, they haven't been used widely in the field of Spoken Dialogue Systems (SDS). In this work, we investigate a novel use of GANs in the field of SDS. Drawing intuition from recent related work, we investigate the use of a form of GANs, the Adversarial Autoencoder (AAE), as we want to explore efficient Belief State (BS) space representations through generative adversarial modeling. We review the difficulties that arise when training a GAN and we propose techniques to improve the training process. In particular, we propose the use of the Wasserstein Adversarial Autoencoder (WAAE), which is based on the Wasserstein loss, and we investigate its effectiveness compared to the baseline AAEs. We also examine the efficiency of the Denoising Adversarial Autoencoder (DAAE) in noisy environments. To evaluate our models, we implemented our algorithms in the PyDial toolkit and we performed several experiments employing two Reinforcement Learning (RL) algorithms, GP-SARSA and LSPI. These two algorithms receive the BS representation from the AAE and optimize the dialogue policy. Our experiments confirm the ability of the generative adversarial modeling to robustly represent the BS space, since the proposed method exhibits state-of-the-art performance, particularly in environments with high levels of noise.

Ανταγωνιστική Μάθηση σε Στατιστικά Συστήματα Διαλόγου

Διαλεκτάκης Γεώργιος

Περίληψη

Τα τελευταία χρόνια, έχει ενταθεί το ενδιαφέρον της ερευνητικής κοινότητας για τα Γενετικά Ανταγωνιστικά Δίκτυα (GANs). Το ενδιαφέρον αυτό βασίζεται στις μεγάλες δυνατότητες που έχουν επιδείξει στη σύνθεση τεχνητών εικόνων, βίντεο και ήχου. Ωστόσο, δεν έχουν χρησιμοποιηθεί ευρέως στον τομέα των Συστημάτων Διαλόγου. Σε αυτή τη διατριβή, ερευνάμε μία καινοτόμα χρήση των GANs στον τομέα των Συστημάτων Διαλόγου. Παρακινούμενοι από μία πρόσφατη σχετική εργασία, προτείνουμε μια νέα χρήση μιας μορφής Γενετικών Ανταγωνιστικών Δικτύων, του Ανταγωνιστικού Αυτόματου Κωδικοποιητή (Adversarial Autoencoder), καθώς θέλουμε να διερευνήσουμε αποδοτικές απεικονίσεις του χώρου των πεποιθήσεων μέσω του μοντέλου ανταγωνιστικής μάθησης. Εξετάζουμε τις δυσκολίες που εμφανίζονται κατά την εκπαίδευση ενός γενετικού ανταγωνιστικού δικτύου (GAN) και προτείνουμε κάποιες τεχνικές για την αντιμετώπισή τους. Συγκεκριμένα, προτείνουμε τη χρήση του Wasserstein Ανταγωνιστικού Αυτόματου Κωδικοποιητή (Autoencoder), ο οποίος βασίζεται στη συνάρτηση απώλειας του Wasserstein, και διερευνάμε την αποτελεσματικότητά του κατά την εκπαίδευση Ανταγωνιστικών Αυτόματων Κωδικοποιητών. Εξετάζουμε επίσης την αποδοτικότητα του Denoising Ανταγωνιστικού Αυτόματου Κωδικοποιητή σε περιβάλλοντα όπου υπάρχει υψηλός θόρυβος. Για να μελετήσουμε την απόδοση των μοντέλων μας, υλοποιήσαμε τους αλγορίθμους μας και εκτελέσαμε διάφορα πειράματα στο εργαλείο PyDial, όπου χρησιμοποιούμε δύο αλγορίθμους Ενισχυτικής Μάθησης, τον GP-SARSA και τον LSPI. Αυτοί οι δύο αλγόριθμοι λαμβάνουν την αναπαράσταση του χώρου των πεποιθήσεων από τον Ανταγωνιστικό Αποκωδικοποιητή και βελτιστοποιούν την πολιτική διαλόγου. Επιβεβαιώνουμε την ικανότητα του ανταγωνιστικού μοντέλου στην ισχυρή αναπαράσταση του χώρου των πεποιθήσεων και δείχνουμε ότι η μέθοδός μας παρουσιάζει παρόμοια και μερικές φορές καλύτερη απόδοση από την τελευταία λέξη της τεχνολογίας, ιδιαίτερα σε περιβάλλοντα με υψηλά επίπεδα θορύβου.

Acknowledgments

I would like to take this opportunity to express appreciation to all the people that stood next to me and supported me throughout the past six years of studying at the Technical University of Crete in the School of Electrical and Computer Engineering. Firstly, I would like to thank Dr.Vasileios Diakoloukas, who guided me throughout this work and was always available, whenever I had questions about my research. Furthermore, I would like to thank my supervisor Dr. Michail G. Lagoudakis, and Prof. Georgios Chalkiadakis, for their useful comments and their time to evaluate this work. Also, I would like to pay particular regards to my uncle, George Em. Karniadakis, who helped me get through some difficulties during my research. Last, but not least, I would like to express my deep gratitude to my family and to Dimitra for providing me with support and constant encouragement during my studies from the first moment to the last one. Thank you.

Contents

Chapter 1

Introduction

Over the past few years, Spoken Dialogue Systems (SDS) [26, 45, 56] have become essential for many industries, such as chatbots[25], tourist information [5], navigation [27], banks [36], and service centers [18], in order to improve their performance and offer better services. There has also been a growing interest in real-world applications, such as Google Assistant, Apple Siri, and Microsoft Cortana, which use voice commands to interact with the user and perform numerous tasks within smart devices. The main component of a Dialogue System is the Dialogue Manager (DM), whose role is to supervise the state of the dialogue and estimate the next action using a Policy Manager (PM). Until now, DMs based on rules have exhibited their effectiveness in several fields. However, they are quite challenging to develop, expensive to scale to real-world problems, sensitive to linguistic faults, and able to perform only in specific domains. This has guided the research to establish Statistical Dialogue Managers who have the benefit of generalization and are capable of operating in continuous space. Statistical Dialogue Managers offer three types of Policy Managers. These are the non-parametric [14], the linear parametric [29, 31], and the non-linear parametric [12, 38, 48] Reinforcement Learning algorithms.

In a benchmark study made by Casanueva et al. in 2017 [6], the non-parametric GP-SARSA came out to exhibit the best performance among all other approaches that were based on Neural Networks. In their analysis, the PMs were optimized by utilizing the summary Belief State (sumBS), which comes from the master Belief State space [55, 54]. Nonetheless, the sumBS vectors are not capable of generalizing and adjusting to different domains as they depend on the domain structure. In addition, they include unnecessary sparse information, which comes in large dimensions.

To overcome the difficulties mentioned above, Lygerakis et al. [34] proposed an innovative technique based on Autoencoders (AEs) in order to generate more robust, efficient and lower-dimensional BS representations that resulted in substantial performance gains, particularly in complex environments. To model the actual distribution that constitutes the sumBS space, they successfully proposed the use of the Variational Autoencoder (VAE). Finally, they considered a noise-robust variation of the autoencoders, the Denoising AE (DAE) and the Denoising Variational Autoencoder (VDAE), respectively, to produce BS vectors that are noise-tolerant especially in environments with high Semantic Error Rate (SER).

1.1 Thesis Contribution

In this work, we examine the effectiveness of Generative Adversarial Networks (GANs) [16] in the context of a dialogue system, as described earlier. Over the last decade, GANs have shown their great potential in the Image Synthesis domain [44, 1, 24, 4]. Recently, there has been a rising interest in applying GANs in the Natural Language Processing (NLP) domain, especially for dialogue generation [30, 43]. However, such techniques have only managed to produce short discrete sequences with small vocabularies that resemble human ones and are considerably far from forming an entire dialogue system. Therefore, we introduce an innovative use of the Adversarial Autoencoder (AAE), [35], which follows the philosophy of generative adversarial modeling, in an effort to explore more efficient BS space representations. Furthermore, we introduce the Wasserstein Adversarial Autoencoder (WAAE), which is an AAE that uses the Wasserstein distance to optimize its parameters [17], as we would like to investigate if it is as effective as in GANs. Finally, we utilize a denoising technique, as proposed in [7], as we consider a noise robust variation of the AAE, the Denoising Adversarial Autoencoder (DAAE) to overcome the challenges introduced by the presence of noise in SDS.

We optimize dialogue policies with two modern RL algorithms, GP-SARSA [14] and Least Squares Policy Iteration (LSPI) [29], both using the representations obtained from the Adversarial Autoencoders. We confirm the efficiency of our proposed technique by conducting several experiments in both common and complex domains, with different noise levels, using the PyDial dialog simulation toolkit [49].

1.2 Thesis Synopsis

In the following section, we summarize the structure of this diploma thesis.

In **Chapter 2**, we present the background of Generative Adversarial Networks and we focus on the training difficulties they introduce. We then discuss the Wasserstein Generative Adversarial Networks (WGANs), which utilize a different cost function based on the Wasserstein distance, providing solutions to the previous problems and leading to more stable training [2]. We also provide background on the field of Spoken Dialogue Systems. We give definitions for the underlying architecture of a SDS and we present two modern Reinforcement Learning algorithms, namely the GP-SARSA and LSPI, which are then applied for our experimental analysis.

In **Chapter 3**, we describe the original Belief State space representation, and we refer to non-linear BS representations based on VDAE [34], which are closely related to our work. We also present the contribution of this thesis.

In **Chapter 4**, we propose a novel use of the Adversarial Autoencoder in BS representation and we study the relationship to Variational Autoencoders. We then present the Wasserstein Adversarial Autoencoder, which uses the Wasserstein loss function as in WGANs in order to examine its effectiveness when training AAEs. Later, we propose a noise-tolerant technique for our Adversarial Autoencoders, the Denoising AAE (DAAE). Finally, we present a concurrent training procedure that optimizes both the parameters of the AAEs and the policies.

In **Chapter 5**, we present our experimental analysis and compare the results of our work with those in Lygerakis' work [33].

In **Chapter 6**, we come to conlusion about this work and we suggest some ideas for further research in the future.

Chapter 2

Background

2.1 Generative Adversarial Neural Networks

2.1.1 Vanilla Generative Adversarial Networks

Generative Adversarial Networks (GANs) have recently attracted the interest among the researchers, especially in the field of image generation. GANs form a class of machine learning schemes in the scope of generative modeling [16]. A GAN composes of two neural networks, the Generator (G) and the Discriminator (D), which battle opposing one another in a game composed of two players (in the sense of Game Theory), while trying to maximize their objective function. The goal of a GAN is to create new artificial data that look like the real data of the training dataset. This could also be useful for dialogue managers, since Generative Networks could be used to generate new BS vectors based on a prior distribution in a procedure that will be described in the next chapter.

To achieve the aforementioned functionality, the generator samples some noise z from a Gaussian or uniform distribution and produces new synthetic data $x = G(z)$. Noise z represents the hidden characteristics of the produced data, so the generator attempts to learn a function that maps from z to x (hopefully, the real data distribution). For this reason, we use a second neural network, a discriminator, which will provide the generator with feedback and direction on what data to generate by learning what features make data real. The discriminator looks at the real training samples as well as the artificially generated ones separately and learns to classify its input as real or fake, i.e., whether the data derive from the training dataset or the generator, respectively. The discriminator's output $D(x)$ indicates the probability of input x being real, $P(input = real\ data)$. For this reason, we use a *sigmoid* activation function in the output layer of the discriminator which produces values between 0 and 1. The model's architecture is presented in Figure 2.1 [19]. If the input to the D comes from the dataset, we want $D(x) = 1$. If it derives from the generator, it has to be zero. Throughout this procedure, D is trained to recognize real data characteristics. The generator, on the other hand, is trained to produce data with $D(x) = 1$. We can optimize the discriminator by backpropagating this goal value back to G, i.e., we optimize G to produce new unseen data towards what the discriminator considers as real. D and G play a two-player minimax game with value function $V(D, G)$, where G wants to minimize V, while D wants to maximize

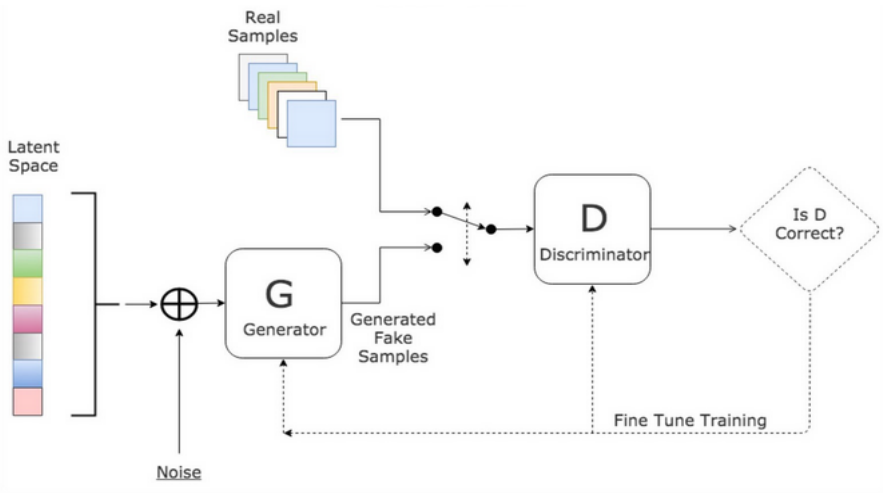


Figure 2.1: Generative Adversarial Network Model [19]

it, as shown in equation 2.1:

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.1}$$

where $x \sim p_{data}$ denotes that sample x is drawn from the distribution p_{data} of the training dataset, while $z \sim p_z(z)$ denotes the samples generated from a prior normal or uniform distribution p_z and serve as the generator's input. The above equation can be simplified as:

$$V(D,G) = \int_x p_{data}(x) log(D(x))\, dx + \int_z p_z(z) log(1 - D(G(z)))\, dz \tag{2.2}$$

Minimizing the objective function (2.1) is identical to reducing the Jensen-Shannon (JS) divergence [16]. The JS-divergence constitutes a technique of calculating the resemblance between two probability distributions, and is formed using the Kullback–Leibler (KL) divergence, as illustrated below.

$$D_{JS}(p\|q) = \frac{1}{2} D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q\|\frac{p+q}{2}) \tag{2.3}$$

where p is the actual data distribution, q is the distribution approximated by the generator and D_{KL} is the KL-divergence.

In practice, the above equation is optimized using the $cross - entropy$ cost function. Once the objective function is defined, G and D are trained mutually using the alternating gradient descent algorithm [40] in the following steps.

- Initialize the parameters of G and D.
- Execute one iteration of gradient descent on D, using both real and artificial data.
- Next, train G for one iteration using back-propagation.
- Repeat from step 2 for as many iterations required to converge.

Eventually, after a number of iterations, they are both optimized. The D recognizes the minor discrepancy between generated and real data, while, the G learns to generate synthetic data that D is unable to distinguish from real ones. Finally, the GAN model should achieve convergence and deliver artificial data very similar to the real ones in the dataset. However, this is not always the case, as GANs depend on the complexity of the training dataset and the specific application, presenting several difficulties as we explain in subsection 2.1.2.

2.1.2 Difficulties when training GANs

Recent applications of Generative Adversarial Networks have proved that they can produce excellent artificial samples [11, 42]. However, the training process of GANs is complex, and there are several difficulties to overcome. Among the ones are the 1) mode collapse, 2) non-convergence, and 3) vanishing gradients:

1. **Mode collapse** describes the failure of the generator to produce data that have adequate diversity and instead it produces data identical to each other. This undesirable behavior can arise when we train a GAN with data with multimodal or very complex distribution. For example, in the MNIST dataset, there are ten distinct classes, from digit '0' to digit '9'. Two different GANs generated the samples shown in Figure 2.2 [37]. The top row shows samples from a well trained GAN that produces all ten classes. The second row is a GAN that creates data from a single class only (the digit '6'), which means that it suffers from mode collapse.

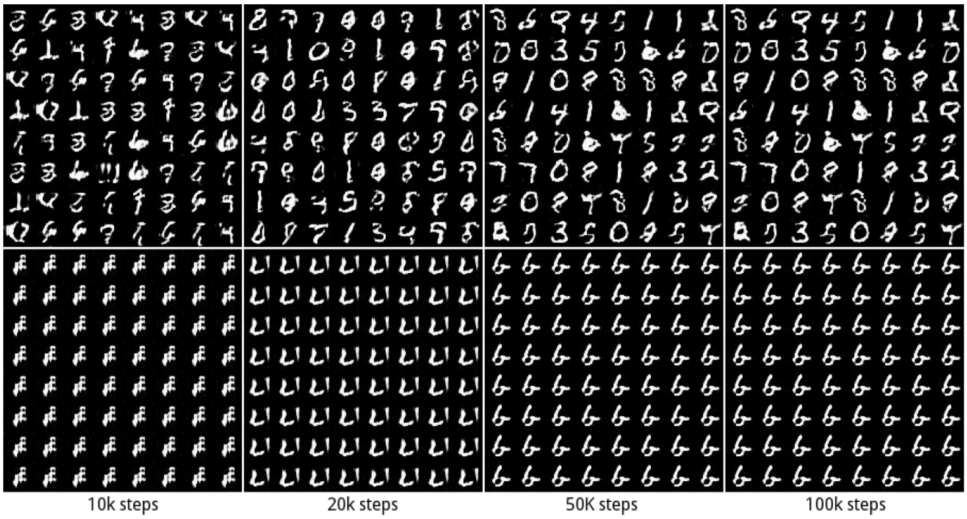


Figure 2.2: Mode collapse in MNIST [37]

2. **Non-Convergence:** As already stated, a GAN consists of two neural networks, a generator G and a discriminator D, where the one tries to challenge the other in order to reach an optimal state. In Game Theory, this optimal state is often called Nash equilibrium, where the best result of a game is when no player has motivation to change his selected strategy after seeing the opponent's decision. The training of GANs aims at reaching a Nash equilibrium

where the generator and the discriminator want to minimize their own cost function $J^{(G)}$ and $J^{(D)}$, respectively. However, a GAN has non-convex cost functions and contains continuous, high-dimensional parameters. To the best of our knowledge there are no feasible algorithms to find Nash equilibria for such cases. For this reason, in order to optimize a GAN, we apply gradient descent methods that seek for the minimum value of both the generator's and the discriminator's loss function simultaneously. Unfortunately, a change to the generator parameters that reduces $J^{(G)}$ can increase $J^{(D)}$, and a change to the discriminator parameters that reduce $J^{(D)}$ can increase $J^{(G)}$. Thus, gradient descent algorithms may fail to reach convergence [15, 47].

3. **Vanishing gradients** happen when Neural Networks are trained through gradient-based methods and backpropagation. Throughout training, the gradient is back-propagated from the last layer of the network to the first one getting progressively lesser. Occasionally, the gradient is so tiny that the first layers of the network learn at a very slow rate or they even cease learning entirely. In the case of GANs, vanishing gradients appear in the generator network when the discriminator is close to optimality. That is because the loss function of an optimal discriminator is very close to 0, generating almost zero gradients, which gives insufficient feedback to the generator. This results in slowing or even stopping the learning of the generator. In fact, as the discriminator improves at distinguishing between artificially generated and real data, it produces progressively smaller gradients resulting in even worse updates to the generator, which makes the GAN model impossible to deliver any training improvement.

2.1.3 Wasserstein Generative Adversarial Networks

As already mentioned, training a Generative Adversarial Network can be quite challenging and may lead to many pitfalls. Even though several methods have been proposed to overcome those problems, such as feature matching, minibatch discrimination, historical averaging, and one-sided label smoothing [47], research has made a lot of effort in introducing new GAN models by adjusting the cost function in order to improve the training procedure. Some examples of such models are the: LSGAN [22], DCGAN [42], BEGAN [21], RSGAN [23]. We will study in detail a cost function which is based on the Wasserstein distance [41].

As already stated, vanilla GANs are trained by minimizing the JS divergence (equation 2.2). In this process, the generator is more difficult to optimize than the discriminator, as the discriminator is a standard classifier, trained to distinguish between just two classes, while the generator needs to learn the underlying distribution of the training data in order to produce artificial ones. If the generator produces high loss at early stages meaning that it is yet unable to produce high-quality synthetic data, then its gradient decreases up to the point where it becomes unable to learn anything, or the whole model becomes unstable as stated by Arjovsky et al [2].

Wasserstein Generative Adversarial Network (WGAN) introduces a new loss function based on Wasserstein distance, which offers a usable gradient on all

parts of the space as it is differentiable almost everywhere and helps the model to learn regardless of the performance of the generator, as explained in [2]. The Wasserstein metric, also called Earth-Mover (EM) distance, is the least possible cost of transfering mass in reforming the distribution P_g to the distribution P_r and is defined as the cost of the cheapest transport plan:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} E_{(x,y)\sim\gamma}[\|x - y\|] \tag{2.4}$$

where P_r and P_g is the real data and the generated data distribution respectively, $\Pi(P_r, P_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are P_r and P_g correspondingly, and inf is the infimum. However, as the above equation is highly intractable, it is possible to simplify the computation by utilizing the Kantorovich-Rubinstein duality [10] as shown below:

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} E_{x\sim P_r}[f(x)] - E_{x\sim P_g}[f(x)] \tag{2.5}$$

where *sup* is the least upper bound and f is a 1-Lipschitz continuous function following the constraint:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|, \qquad \forall\ x_1, x_2 \in \mathbb{R} \tag{2.6}$$

The 1-Lipschitz continuity is important as it limits how fast function f can change. If $f(x_1)$ and $f(x_2)$ are connected to each other with a line, its slope has an absolute value always smaller than 1. Therefore, in Deep learning, Lipschitz continuity restricts the gradients and helps diminish exploding or vanishing gradients.

To compute the Wasserstein distance, it is essential to acquire an 1-Lipschitz function. This can be achieved by deploying a deep neural network. In fact, this network is very analogous to the GAN's discriminator. The difference is that it outputs a scalar score in $(-\infty, \infty)$ rather than a probability, which can be viewed as a measurement of the realness of the input data. To do this, a *linear* activation function is used instead of the *sigmoid* in the output layer. Finally, the discriminator's name is changed to critic to express its new functionality.

The significant difference between vanilla GANs and WGANs arises only on the cost function which transforms from Equation 2.1 to:

$$\min_G \max_{D\in L} E_{x\sim P_r}[D(x)] - E_{\tilde{x}\sim P_g}[D(\tilde{x})] \tag{2.7}$$

where L denotes the set of 1-Lipschitz functions and P_r is the real data distribution and P_g is the generated data distribution defined by $\tilde{x} = G(z),\ z \sim p(z)$. Under these circumstances, if the critic is optimal, minimizing the value function towards the parameters of the generator diminishes $W(P_r, P_g)$ and the generator output becomes closer to the real data distribution.

Finally, f needs to be a 1-Lipschitz function. To impose this restriction, WGAN uses a simplistic clipping method to limit the maximum value of the weights in f, i.e., the critic weights are constrained in a range of values defined by the hyperparameters c in the method *clip* shown below. Moreover, instead of the

Adam, the *RMSprop* optimizer is used to update the weights of the discriminator.

$$w \longleftarrow w + \alpha \cdot RMSProp(w, g_w)$$
$$w \longleftarrow clip(w, -c, c)$$

In algorithm 1 [2], we show the complete training procedure of WGANs, where w_0 and θ_0 are the randomly initialized weights of the critic and generator, respectively.

Algorithm 1 WGAN proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

2.2 Dialogue Management

A Spoken Dialogue System (SDS) [57] is composed of a Spoken Language Understanding component (SLU), a Dialogue Manager (DM), which includes the Belief State Tracker and the Policy Manager (PM), and a Natural Language Generation component (NLG) as shown in figure 2.3.

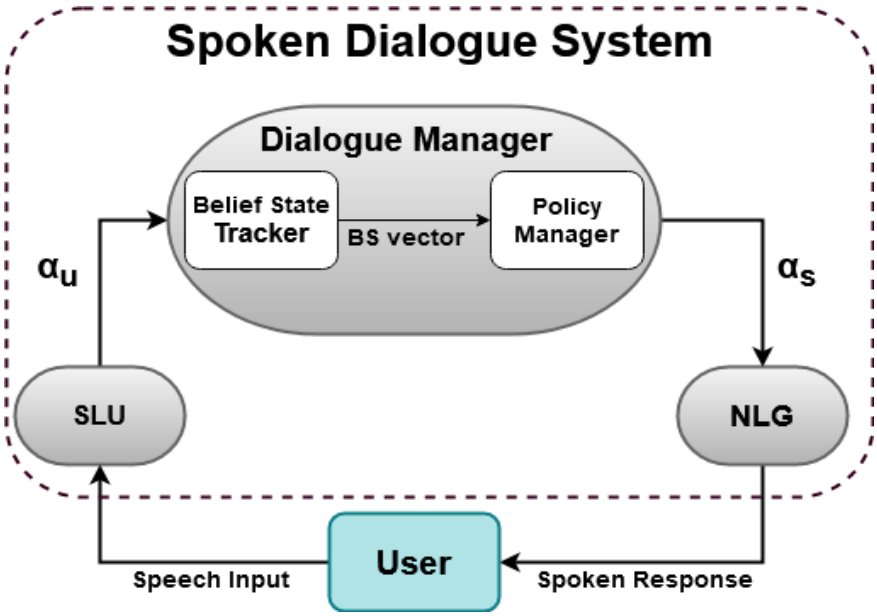


Figure 2.3: Spoken Dialogue System Architecture [33]

Consecutive interactions between the user and the system form a dialogue which terminates when the user's goal is fulfilled. In every turn, the SDS receives voice input from the user. Initially, the user's speech is converted to text by a speech recognizer inside the SLU module. It is then translated into a semantic representation, which has the scheme of a user dialogue action a_u, by the natural language understanding component. The belief state tracker in the DM utilizes a_u to determine the ongoing dialogue's belief state and supplies the PM with a BS vector, which is then used to decide the next action of the system via a decision rule $a_s = \pi(b_s)$, where π is the current policy. Finally, the responsibility of the NLG component is to convert the system action back to a spoken response in two steps. First, written language is obtained from the system act a_s and then the system replies to the user in spoken language with the help of a text-to-speech module.

2.2.1 MDPs and POMDPs in dialogue

Markov Decision Processes (MDPs) [3] can help us model how a dialogue evolves. MDPs adopt the Markov property, where the next state does not depend on any of the previous states and actions, but depends only on the current state and the decision maker's action. More specifically, an MDP is defined as a $4-tuple$:

$$(S, A, T_a, R_a) \tag{2.8}$$

where S is a finite set of states, A is a finite set of actions, $T_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time step t will lead to state s' at time $t + 1$ and $R_a(s, s')$ is the immediate reward received after transitioning from state s to state s', due to action a.

However, in real-life SDS, the state s_t is partially observable due to the uncertainty in the interpretation of user utterances. This uncertainty is caused by the speech recognition errors, the ambiguity of the user's actual intention, and the errors at the semantic extraction. Therefore, we can model the dialogue manager as a Partially Observable Markov Decision Process (POMDP) [57, 14].

A discrete-time POMDP represents the interaction of an agent with its environment. In the case of an SDS, the agent refers to the Dialogue Manager. Usually, the following $7-tuple$ is able to model a POMDP [26]:

$$(S, A, T, R, \Omega, O, \gamma) \tag{2.9}$$

where S is a set of states, A is a set of actions, T is a set of conditional transition probabilities between states, $R : S \times A \rightarrow \mathbb{R}$ is the reward function, Ω is a set of observations, O is a set of conditional observation probabilities, and $\gamma \in [0, 1]$ is the discount factor. In a POMDP (Figure 2.4 [57]), at each time-step, the dialogue is in some partially-observable state s_t. The agent uses the derived policy π to perform an action $a_t \in A$ and transition to a new state s_{t+1} with probability $T(s_{t+1} \mid s_t, a_t)$. Finally, a reward r_t, equal to $R(s_t, a_t)$, is received by the agent while taking an observation $o_{t+1} \in \Omega$ which relies on the dialogue's new state, s_{t+1} with probability $O(o_{t+1} \mid s_{t+1})$. Then the process repeats.

However, as the ongoing state of the dialogue, s_t, is not known precisely, we keep a distribution over all probable states, named belief state $\boldsymbol{b} \in B : \mathbb{R}^{|S|}$. By keeping a belief state vector, the DM is able to efficiently seek after all feasible dialogue paths concurrently, choosing the next action depending on the probability distribution across all states, rather than the most probable one. The probability of the ongoing most probable state decreases, whenever the user indicates a problem, and the focus shifts to a different state. This permits robust dialogue policies to be incorporated in a straightforward homogeneous mapping from belief state to action.

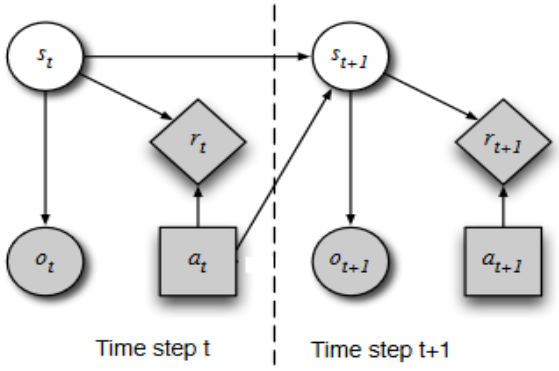


Figure 2.4: POMDP. Clear circles indicate hidden states; shaded circles are observations; squares are system actions; diamonds are real-valued rewards; and arrows show causality. [57]

At each time step t, the goal of the Policy Manager is to acquire a policy π of actions a, $a_t = \pi(\boldsymbol{b_t})$, in order to maximize the discounted expected reward beginning from the primary belief $\boldsymbol{b_0}$:

$$V^{\pi}(\boldsymbol{b_0}) = \sum_{t=0}^{\infty} \gamma^t E[R(\boldsymbol{b_t}, a_t)|\boldsymbol{b_0}, \pi] \tag{2.10}$$

Then, we receive the optimal policy π^* from:

$$\pi^* = \arg\max_{\pi} V^{\pi}(\boldsymbol{b_0}) \tag{2.11}$$

We can acquire π^* for every belief state $\boldsymbol{b}$ by receiving the value function V^* of an optimal policy π^* for each b. By solving the Bellman equation, we are able to acquire V*:

$$V^*(\boldsymbol{b}) = \max_{a \in A} \left[r(\boldsymbol{b}, a) + \gamma \sum_{o \in O} Pr(o|\boldsymbol{b}, a) V^*(T(\boldsymbol{b}, a, o)) \right] \tag{2.12}$$

where T is the transition function of the belief state.

At this moment, we could use Reinforcement Learning algorithms to estimate optimal policies by associating the belief state vector and the equivalent action acquired from the policy component with the environment reward.

Unfortunately, in a real-world dialogue system, the number of states, actions and observations can easily be more than 10^{10}. This complexity introduces barriers in finding solutions to the above equations. Moreover, the process to acquire the transition function T forms an intractable problem and as a result we are unable to utilize value or policy iteration or dynamic programming to estimate an optimal policy. In the following section, we describe how we can overcome these difficulties by utilizing modern *RL* algorithms.

2.3 Reinforcement Learning

To address the problem of computing the transition function T, as defined in 2.9, we need to take advantage of model-free Reinforcement Learning algorithms which estimate an optimal policy without using or considering the dynamics (transition and reward functions) of the environment in contrast to model-based RL algorithms which use the transition function T in order to do so. Q-learning [52] belongs to this family of RL algorithms, which calculates the Q-function, $Q : B \times A \rightarrow \mathbb{R}$, instead of the value function $V : B \rightarrow \mathbb{R}$ and updates Q at each time step t in the following way:

$$Q^{new}(\boldsymbol{b_t}, a_t) \leftarrow Q(\boldsymbol{b_t}, a_t) + \mathrm{a} \left(r_t + \gamma \max_{a_{t+1}} Q(\boldsymbol{b_{t+1}}, a_{t+1}) - Q(\boldsymbol{b_t}, a_t) \right) \tag{2.13}$$

where $Q(\boldsymbol{b_t}, a_t)$ denotes the previous value, $Q(\boldsymbol{b_{t+1}}, a_{t+1})$ is the expected optimal future reward, a is the learning rate, r_t is the reward received by the agent and γ is the discount factor. The Q-function models the quality of an action that is taken to

move to a state rather than determining the possible value of the state being moved to and only depends on the $4-tuple$:

$$(\boldsymbol{b}, a, r, \boldsymbol{b'}) \tag{2.14}$$

where $(\boldsymbol{b'}, \boldsymbol{b}) \in B$ indicate the current and previous belief states correspondingly, a comes from the set of actions A and r is a real number.

Q-learning belongs to the family of off-policy algorithms, determining the next action greedily, i.e., the action that yields the maximum reward, rather than pursuing a specific policy π. Thus, Q-learning performs optimization on-line during its direct interaction with the user.

State-Action-Reward-State-Action ($SARSA$) is another RL model-free algorithm [46]. $SARSA$ is similar to the aforementioned Q-Learning algorithm; however, it is an on-policy algorithm using the action executed by the ongoing policy π to determine the Q-value, as shown in the following equation.

$$Q^{new}(\boldsymbol{b_t}, a_t) \leftarrow Q(\boldsymbol{b_t}, a_t) + a\left(r_t + \gamma Q(\boldsymbol{b_{t+1}}, a_{t+1}) - Q(\boldsymbol{b_t}, a_t)\right) \tag{2.15}$$

Nonetheless, Q-learning and $SARSA$ are not suitable for real-world problems, as they utilize matrices to save the value function data. Therefore, they are not able to handle massive BS and action spaces. In order to surmount this barrier, we utilize function approximation methods to estimate the Q-function. Some of the most distinguished function approximators that have been examined for Dialogue Managers until now, are Gaussian Processes (GP) [14], Least-Squares Policy Iteration ($LSPI$) [29], and Deep Neural Networks ($DNNs$) [38].

2.3.1 GP-SARSA

A Gaussian Process (GP) consists of a group of random variables indexed by time or space whose every finite linear combination is defined by a normal distribution, i.e., they are represented by a multivariate Gaussian distribution. A GP is described by the joint distribution of all those infinite random variables, thereby, it is a distribution over functions with a continuous domain. Therefore, a GP involves an infinite number of parameters, making it a non-parametric function approximator. In this procedure, we begin from the prior knowledge we possess, and we utilize the Bayes' rule in order to update the distribution of function by observing training data [14].

A Gaussian Process is defined by a mean and a kernel function, which represents relationships that exist between various segments of the BS space and is essential for obtaining reasonable estimations with just a few observations. The GP-Sarsa algorithm uses Gaussian Processes to define a policy for dialogue systems by modeling the Q-function as a GP with zero mean [57]:

$$Q(\boldsymbol{b}, a) \sim \boldsymbol{GP}(0, k((\boldsymbol{b}, a), (\boldsymbol{b}, a))). \tag{2.16}$$

where $k((\boldsymbol{b}, a), (\boldsymbol{b}, a))$ is a kernel, as mentioned earlier, that describes the relationships in belief-action space $(\boldsymbol{b}, a)$. It has been shown that GP-Sarsa can optimize dialogue policies sooner than a regular RL algorithm [14], by training a dialogue policy in immediate communication with human users successfully while handling arbitrary high dimensional states [13]. Finally, the GP-Sarsa algorithm is obtained by replacing the Q-function correction in the right-hand side of Eq 2.13 with the update of the Q-function approximation based on GP.

2.3.2 Least-Squares Policy Iteration

The Least Squares Policy Iteration algorithm [29] resides in the group of approximate policy iteration Reinforcement Learning algorithms, able to learn decision policies either from a corpus generated with some other dialog manager, or learn while it is controlling the dialog. LSPI is sample-efficient, making maximal use of data. Moreover, compared to other popular learning methods that support off-policy and on-policy learning, such as Q-Learning, LSPI learns better policies with less data. The algorithm initiates by approximating the Q-function with unrestricted weights w:

$$\tilde{Q}^{\pi}(\boldsymbol{b}, a; w) = \sum_{i=1}^{k} \phi_i(\boldsymbol{b}, a) w_i \tag{2.17}$$

The magnitude of a weight w_i indicates the contribution of its feature $\phi_i(\boldsymbol{b}, a)$ to the Q-function. The basis functions ϕ_i are non-linear functions of $\boldsymbol{b}$ and a, whose size is determined by the specific application. Next, the algorithm advances to policy optimization, selecting greedily the action that maximizes Q^{π_m} at each iteration m (Eq. 2.18) and policy assessment (Algorithm 1), where we utilize the Least-Squares Temporal Difference Q-function (LSTDQ) algorithm in order to estimate the Q-function.

$$\pi_{m+1}(\boldsymbol{b}) = \arg\max_{a \in A} \tilde{Q}^{\pi_m}(\boldsymbol{b}, a) \tag{2.18}$$

In Algorithm 2 [29], we make use of the symbol s to denote the dialogue state i.e. the BS, rather than the symbol $\boldsymbol{b}$. This is done so there is no complication of the notation between the BS vector and the bias. However, in reality, the LSTDQ algorithm uses samples of the form $(\boldsymbol{b}, a, r, \boldsymbol{b'})$.

However, the LSTDQ algorithm requires the calculation of the matrix $\widetilde{\boldsymbol{A}}^{-1}$, which depends on the number of basis functions k with a complexity of $(O(k^3))$, leading to poor system behavior, particularly when the frequent update of equation (2.17) is needed. An optimized vesion of LSTDQ, namely LSTDQ-OPT, is able to directly update the A matrix, which we denote as $\boldsymbol{B}$ in Algorithm 3 [29]. By making such a modification, we are able to update our linear system in a quick and cost-effective way, having a dependence on the number of basis functions k with a complexity of only $(O(k^2))$. Eventually, to determine an optimal policy, we utilize the LSPI algorithm as summarized in Algorithm 4 [29].

Algorithm 2 LSTDQ algorithm with a model - Learns the approximate Q of a fixed policy

function LSTDQ(D, k, ϕ, γ, π) ▷ Learns $\hat{Q}^{\pi}$ from samples
// D : Set of samples (s, a, r, s')
// k : number of basis functions
// ϕ : Basis functions
// γ : Discount factor
// π : Policy whose value function we are seeking at each time step

    $\tilde{\boldsymbol{A}} \leftarrow \boldsymbol{0}$ ▷ Weights: $(k \times k) matrix$
    $\tilde{\boldsymbol{b}} \leftarrow \boldsymbol{0}$ ▷ Bias: $(k \times 1) vector$

    for each $(s, a, r, s') \in \mathcal{D}$
        $\tilde{\boldsymbol{A}} \leftarrow \tilde{\boldsymbol{A}} + \phi(s, a)\phi\left(\phi(s, a) - \gamma\phi(s', \pi(s'))\right)^T$
        $\tilde{\boldsymbol{b}} \leftarrow \tilde{\boldsymbol{b}} + \phi(s, a)r$
    $\tilde{\omega}^{\pi} \leftarrow \tilde{\boldsymbol{A}}^{-1}\tilde{\boldsymbol{b}}$

    return $\tilde{\omega}^{\pi}$

Algorithm 3 LSTDQ-OPT algorithm - Direct update on $\tilde{\boldsymbol{A}^{-1}}$

function LSTDQ(D, k, ϕ, γ, π) ▷ Learns $\hat{Q}^{\pi}$ from samples
// D : Set of samples (s, a, r, s')
// k : number of basis functions
// ϕ : Basis functions
// γ : Discount factor
// π : Policy whose value function we are seeking at each time step

    $\tilde{\boldsymbol{B}} \leftarrow \frac{1}{\delta}\boldsymbol{I}$ ▷ $\tilde{\boldsymbol{A}^{-1}}$: $(k \times k) matrix$
    $\tilde{\boldsymbol{b}} \leftarrow \boldsymbol{0}$ ▷ Bias: $(k \times 1) vector$

    for each $(s, a, r, s') \in \mathcal{D}$
        $\boldsymbol{B} \leftarrow \boldsymbol{B} - \frac{B\phi(s,a)(\phi(s,a)-\gamma\phi(s',\pi(s')))^T B}{1+(\phi(s,a)-\gamma\phi(s',\pi(s')))^T B\phi(s,a)}$
        $\tilde{\boldsymbol{b}} \leftarrow \tilde{\boldsymbol{b}} + \phi(s, a)r$
    $\tilde{\omega}^{\pi} \leftarrow \tilde{\boldsymbol{B}}\tilde{\boldsymbol{b}}$

    return $\tilde{\omega}^{\pi}$

Algorithm 4 LSPI algorithm - Iteration between Policy Improvement and Policy Iteration

function LSPI$(D, k, \phi, \gamma, \epsilon, \pi_0)$ ▷ Learns a policy from samples
// D : Set of samples (s, a, r, s')
// k : number of basis functions
// ϕ : Basis functions
// γ : Discount factor
// ϵ : Stopping Criterion
// π_0 : Initial policy, given as w_0

    $\pi' \leftarrow \pi_0$ ▷ $w' \leftarrow w_0$

    repeat
        $\pi \leftarrow \pi'$ ▷ $w \leftarrow w'$
        $\pi' \leftarrow LSTDQ(D, k, \phi, \gamma, \pi)$ ▷ $w' \leftarrow LSTDQ(D, k, \phi, \gamma, \pi)$
    until $(\pi \approx \pi')$ ▷ **until** $||w - w'|| < \epsilon$

    return π ▷ **return** π

In this thesis, we are using the block-basis functions, which is among the most straightforward state-action encodings. Considering a dialog domain of n distinct dialog actions $A = (a_1, a_2, ..., a_n)$, we can formulate the block-basis feature as a sequence of n block vectors, each having the dimensions of the Belief State vector. In this context, we obtain the state-action representation by the activation of an individual feature-vector block that corresponds to the chosen action, while the rest of the blocks in the feature-vector take the value of zero. Particularly, if we represent the BS with a d-dimensional vector $s = (s_1, s_2, ..., s_d)$, the block-basis feature for the state s and a corresponding active action a is formed as follows:

$$\phi(\boldsymbol{s}, a) = [c_1\boldsymbol{s}, c_2\boldsymbol{s}, ..., c_n\boldsymbol{s}]^T \tag{2.19}$$

where $c_i, i = 1, ..., n$ is a binary constant determined as:

$$c_i = \begin{cases} 1, & \text{if } a_i = a. \\ 0, & \text{otherwise.} \end{cases} \tag{2.20}$$

Even though the above process can be a simple, yet efficient representation in the context of a dialog system, since it accurately specifies the contribution of each action, it also has several drawbacks. For example, even an average in size action space can lead to sparse and high dimensional block-basis features for a common BS vector. This results in high computational loads throughout the policy training procedure, making the usage of real-world action spaces impractical.

Chapter 3

Belief State Space Representation

3.1 Full Belief State Space

In this diploma thesis, we consider "slot-filling" dialogs. The states of such a dialog, consist of M slots, where slot j receives one out of M_j values. The dialog is initiated by the user's intention, and in this context, every slot name is expected to take a specific value. The system then aims to fulfill the user's goal correctly. For instance, the following are two slot name-value examples which represent the user input on a dialogue state of the Cambridge Restaurants (CR) and Laptops11 (LAP11) domains respectively. Specifically, in the first example, the next action of the system is to inform the user that the price of the restaurant is expensive, the food is chinese, the restaurant is located at the center of the town and its name is "dojo noodle bar". In the second example, the system informs the user that the processor of the laptop is the "inte core i7", its price is moderate, its system memory is 8 gb and its warranty is 3 years.

- inform(pricerange="expensive", food="chinese", area="centre",..., name="dojo noodle bar") (0.18)
- inform(processorclass="intel core i7", pricerange="moderate", sysmemory= "8 gb",..., warranty="3 year international")(0.05)
- ...

The parenthesis on the right side defines the probability of being in that state, indicating a distribution among all potential states is maintained, as discussed earlier in subsection 2.2.1. Based on the above examples, we can easily deduce that the total amount of possible dialogue states depends exponentially on the total number of the distinct values that each feature can take. As an example, we can think of a dialogue with 5 features in the ontology of the domain, and each feature can take one of $N = 10$ values. In such a situation, the DM is operating on a set of $N^5 = 10^5$ states, preventing RL algorithms from obtaining good policies in a reasonable time.

3.2 Summary Space

In [55, 54], Williams and Young proposed a mechanism, namely the summary space that permitted the full BS space (master space) to be described in a reduced di-

mension and was able to make policy optimization tractable. The summary space constitutes a summarization of the full BS space, and allows the Policy Manager to act exclusively on this BS subset, since most reasonable system responses will focus on just the most likely states, and the entire set of states is never visited.

The formation of the summary space is made by dividing the fullBS space into two sets, S_u^w, which is the user's intention state space, and S_d^w, which is the dialogue's history space [58]. Both sets are decomposed into W slots, and w denotes each slot in the set of W slots. S_u^w indicates the set of values for slot w, and S_d^w indicates the set of possible dialog histories for slot w. The outcome is the formation of two summary space groups $\hat{S}_u^w = \{best, rest\}$ and $\hat{S}_d^w = S_d^w$. To project the fullBS space to the summary one, we make each belief element $\hat{b}(\hat{s}_u^w = best)$ equal to the probability mass of the most probable user intention in slot w, $\hat{b}(\hat{s}_u^w = rest)$ equal to the remaining probability mass of all the rest of the user intentions in slot w, and the dialog history in summary space $\hat{b}(\hat{s}_d^w)$ equal to the dialog history in fullBS space $\hat{b}(\hat{s}_d^w)$.

Consequently, the summary BS vector, inside PyDial, is designed as a dictionary which consists of three sub-dictionaries as shown in the following example:

- domain-dependent userActs:
  - inform(price =“moderate”, utility=“dontcare”)
  - affirm(warranty = “3 year international”)
- beliefs:
  - domain-independent: “method”, “discourseAct”
  - domain-dependent:
    - “warranty”, “processorclass”, “sysmemory” ... (Laptops 11 domain)
    - “food”, “area”, ... (Cambridge Restaurants domain)
- domain-independent features
  - “offerHappened”
  - “informinfo”
  - “lastInformedVenue”
  - “informedVenueSinceNone”
  - “lastActionInformNone”

The domain's ontology determines the values that each of the above slots can receive. In Pydial, the Focus Tracker [20] is responsible for updating the summary BS vector by assigning a probability in each slot. This probability depends on the assumption of the SLU module. Eventually, the above dictionary is compressed into a vector of probabilities, whose length depends on the domain, as shown in the following Tables:

1	2	3	...	636
discourseAct: "hello"	area: "centre"	offerHappened: "false"	...	inform (food="italian", name="dojo noodle bar")
0.035	**0.962**	**0.000**	**...**	**1.000**

Table 2.1: Example of summary BS vector in San Francisco Restaurants domain.

1	2	3	...	257
discourseAct: "ack"	utility: "touchscreen"	offerHappened: "false"	...	inform (warranty="3 year european", platform="windows 8")
1.000	**0.996**	**0.085**	**...**	**1.000**

Table 2.2: Example of summary BS vector in Laptops11 domain.

However, even the received summary BS vector depends on the domain, comes in high dimensions, is sparse, and contains unnecessary information. Those features of the summary BS space refrain most Policy Managers from achieving optimal policies in a reasonable time. Therefore, an efficient and compressed representation of the summary BS space becomes essential.

3.3 Related Work

To alleviate the problems mentioned above, alternative techniques for BS representation have been developed. Such an alternative is the Domain-Independent Parameterization (DIP) [51], which generates BS vectors using domain-independent features of the dialogue. In their work, they maintain a fixed-size dimensional space, which can be more compressed than the summary BS one, so all the domains can be mapped onto a common belief space. However, their method is based on hand-crafted features that do not include useful domain-specific information, resulting in representations that are still sensitive to noise and redundant.

Another alternative is the BinLin/BinAux BS [28], which uses little information from the summary space to obtain very low-dimensional BS representations. Nevertheless, their proposed method highly depends on the specific domain and presents significant performance instabilities. In [31], an automatic technique for feature selection is proposed to obtain compact BS representations. The main drawback here is that their method cannot be generalized to different policy algorithms, as it is highly associated with specific Reinforcement Learning algorithms, such as Least-Squares Policy Iteration.

Current state-of-the-art [34, 8] includes a non-linear BS representation technique that is based on deep Autoencoders. Specifically, they introduced a novel use of a feed forward Autoencoder (AE), the Denoising Autoencoder (DAE), and the Variational Denoising Autoencoder (VDAE) to receive more compact and noise-robust BS representations that were utilized by two RL policy algorithms, GP-SARSA and

Least-Squares Policy Iteration to achieve top performance, even in noisy environments.

3.4 Problem Statement

In this thesis, taking advantage of the excellent ability of the VAE/VDAE to produce noise-robust, compact meaningful BS representations, we propose a novel use of the Adversarial Autoencoder (AAE) to obtain robust BS space representations in a similar technique to Variational Autoencoders. However, while VAE uses the Kullback–Leibler divergence to enforce a prior distribution on the latent code of the autoencoder, the AAE follows an adversarial training process, as in GANs, in order to match the posterior distribution of the autoencoder's hidden representation to a prior distribution of our desire. We also propose the Wasserstein Adversarial Autoencoder (WAAE), which utilizes the Wasserstein loss, to investigate the efficiency of such a technique when training AAEs.

Chapter 4

Belief State Representation with Adversarial Autoencoders

4.1 Vanilla Adversarial Autoencoder

Adversarial Autoencoders (AAE), proposed by Makhzani et al. [35], were initially designed for semi-supervised classification, unsupervised clustering and disentangling style in the image domain. In this thesis, we use AAEs for dimensionality reduction. Our aim is to extract meaningful features of the sumBS vector which will serve as input to the policy manager.

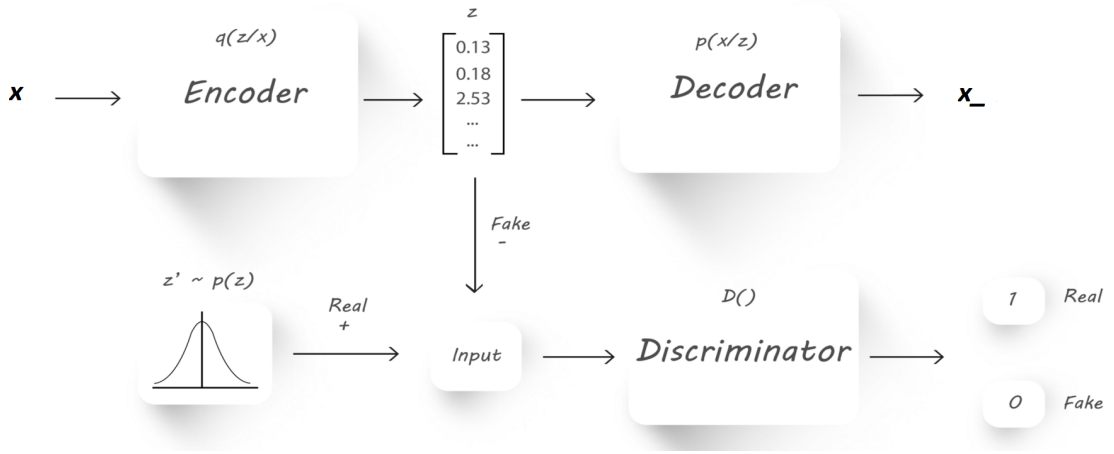


Figure 4.1: Adversarial Autoencoder architecture [39]

AAE is a generative autoencoder based on the framework of GANs, as discussed in Section 2.1.1. AAEs perform variational inference by matching the posterior distribution of the autoencoder's latent code to a desirable prior distribution, which guarantees that generating from any part of prior space leads to meaningful samples. Therefore, the AAE's decoder learns a deep generative model that connects the prior to the data distribution. An adversarial network, which we refer to as the discriminator (D), is added on top of the autoencoder's latent code vector, which will guide the encoder's output to meet the prior distribution. The architecture of the AAE is illustrated in figure 4.1 [39], where $q(z|x)$ denotes the output of the

encoder for an input x, z is the latent encoding drawn from $q(z|x)$, z' is the real input of the discriminator with the prior distribution, $p(x|z)$ is the decoder output given z, and $x\_$ is the reconstructed input. In this work, $x \in \mathbb{R}^{d_x \times 1}$ is the summary BS vector as discussed in Chapter 2 whose size d_x relies on the selected domain, z is a fixed-size, lower-dimensional projection of the sumBS vector and $x\_$ is the reconstructed sumBS vector. The encoder, the decoder and the discriminator are all defined as feed-forward dense Neural Networks composed of several layers.

We will use the encoder ($q(z|x)$) as our generator to ensure that the aggregated posterior distribution can trick the discriminator to believe that the latent code $q(z)$ arises from the real prior distribution $p(z)$. Then the discriminator indicates whether the samples come from the real distribution ($p(z)$) or the encoder (z), and the decoder ($p(x|z)$) to get back the original sumBS vector.

The training of an AAE is performed in two steps, the reconstruction, and the regularization.

- **The reconstruction** is similar to the training of a vanilla autoencoder. Both the encoder and the decoder are trained so as to decrease the reconstruction loss, defined as the mean squared error between the encoder's input x and the decoder's output $x\_$ as shown in the following equation:

$$J(x, x\_) = E[\|x - x\_\|^2] \tag{4.1}$$

  While the discriminator is not taking part in this phase, we pass the sumBS vector as input to the encoder, which will give us latent encodings, which we will provide to the decoder to get back the original sumBS vector. The back-propagation is performed through both the encoder and the decoder weights so that the reconstruction loss will be reduced.

- **The regularization** is when the discriminator and the encoder (generator) are optimized, similarly to the training of a GAN. First, the discriminator is trained to classify the encoder's output (z) and samples z' that come from the prior distribution p_z. The discriminator should give an output close to 1, if its input comes from the prior distribution $p(z)$, while it should give an output close to 0, if its input is the latent encodings produced by the encoder. Intuitively, both the encoder output and the vector with the prior distribution should be of the same shape and size. The next step is to force the encoder to output encodings following a predefined distribution. To accomplish this, we attach the discriminator to the output of the encoder. Next, we fix the discriminator weights, making them untrainable, and set its output to have the value of 1. We feed the encoder with sumBS vectors and identify the output of the discriminator, which is applied to determine the loss using the cross-entropy cost function (Equation 4.2). At this point, the backpropagation is only performed through the encoder weights, which causes the encoder to learn the required distribution by looking at the discriminator weights.

$$C = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)] \tag{4.2}$$

  where n is the batch size, the sum is over all training inputs, x, y is the

corresponding desired output of the discriminator and a is the prediction of the discriminator.

An important decision to make when considering an AAE is the choice of the encoder $q(z|x)$ to be used. Specifically, the encoder can be deterministic, where $q(z|x)$ is considered to be a deterministic function of x. Therefore, the encoder network is identical to the one of a regular AE, where the stochasticity in $q(z)$ comes only from the data distribution, $p_d(x)$. Alternatively a Gaussian posterior can be used as the encoder of the network. In this case, the input of the encoder is mapped to a Normal distribution, whose mean and variance is estimated by the encoder: $z_i \sim N(\mu_i(x), \sigma_i(x))$, similar to the Variational Autoencoder. Thus, both the randomness of the normal distribution at the encoder's output and the data-distribution contribute to the stochasticity of the encoder. Then, we can apply the same re-parameterization trick proposed by Kingma and Welling, [9], as in Variational Autoencoders, to back-propagate through the encoder. Specifically, the encoder's output z is a stochastic node, and backpropagation does not work on random nodes. For this reason, in order to be able to apply backpropagation through the encoder, we can approximate the encoder's output z with an auxiliary normally distributed parameter ϵ that allows the node to be deterministic and permits backpropagation through the network [9].

Based on the selection of the encoder, different types of AAEs will emerge each one having unique training dynamics [35]. When using a deterministic encoder, the network matches the posterior $q(z)$ to the prior $p(z)$ by only utilizing the stochasticity of the data distribution. However, as the data distribution is defined by the training set, this deterministic procedure may create a $q(z)$ that is not very smooth. On the other hand, using a Gaussian posterior could help the AAE in the adversarial regularization phase, as it provides the encoder with additional sources of stochasticity which results in smoother $q(z)$.

In this work, we use the Gaussian posterior for our encoders, as it provides more stable training and helps in matching the desirable prior distribution better than the deterministic encoder. We also maintain the log values of σ, rather than just σ, which offers consistency throughout the training process and makes calculations simpler.

4.1.1 Variational Autoencoder and its relationship to AAE

Variational Autoencoder (VAE) forms a generative model composed of an encoder and a decoder trained to minimize the reconstruction error between the output of the decoder and the input of the encoder. However, it differs from standard autoencoders, as it performs a regularization technique in the latent encoding. Specifically, every input of the VAE is encoded as a distribution over the latent space rather than a single point. Next, a point from the latent space is sampled from that distribution, which is then fed to the decoder to compute the reconstruction error and backpropagate it through the network. The architecture of the VAE is illustrated in Figure 4.2 [53].

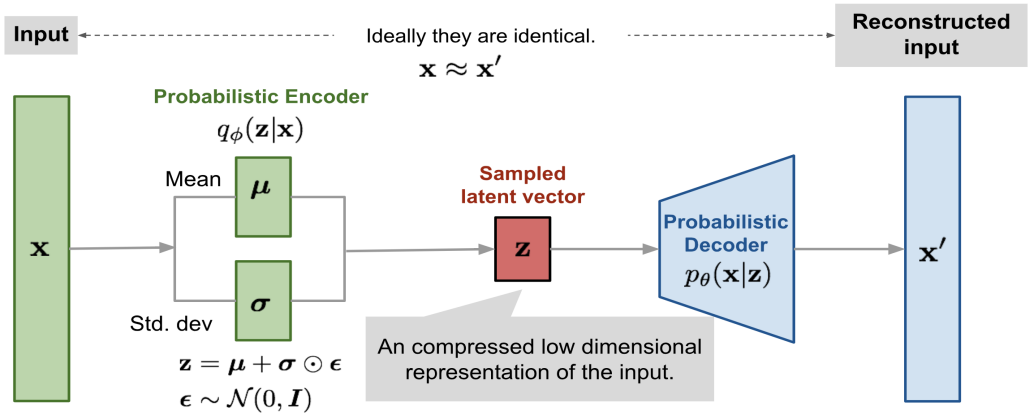


Figure 4.2: Variational Autoencoder architecture [53]

The encoded distributions are chosen to be normal, described by a vector of means $\boldsymbol{\mu}$ and a vector of standard deviations $\boldsymbol{\sigma}$. The vector of means controls the center around which an input is encoded, and the standard deviation controls the space, how far from the mean the encoding can be. VAE ideally wants to produce encodings that are as close as possible to each other while still remaining separated. In this way, it allows smooth interference and enables the generation of new samples. To achieve the regularization functionality, VAEs incorporate the Kullback–Leibler [9] divergence into their loss function. The KL divergence is applied between the distribution produced by the encoder and a standard Gaussian, in order to force the encoder to create encodings that follow a normal distribution. The complete cost function of VAEs is shown below:

$$\begin{aligned} E_{x\sim p_d(x)}[-log\ p(x)] &< E_x[E_{q(z|x)}[-log(p(x|z)]] + E_x[\boldsymbol{KL}(q(z|x)||p(z))] \\ &= E_x[E_{q(z|x)}[-log(p(x|z)]] - E_x[H(q(z|x))] + E_{q(z)}[-log\ p(z)] \\ = E_x[E_{q(z|x)}[-log(p(x|z)]] - E_x[\sum_i log\sigma_i(x)] &+ E_q(z)[-log\ p(z)] + const. \\ &= Reconstruction - Entropy + CrossEntropy(q(z), p(z)) \end{aligned} \tag{4.3}$$

where the aggregated posterior $q(z)$ is sampled from the output of the encoder $q(z|x)$, and $p(z)$ is usually a standard Gaussian distribution.

The Adversarial Autoencoders share common features with the VAEs. They both use the reconstruction loss to compute the difference between the input sumBS vectors and the decoded ones. In our work, AAEs use a probabilistic encoder similar to VAEs, to produce encodings that follow a normal distribution. However, the major difference is the method used to enforce this distribution on the latent encoding. In AAEs, the KL divergence is replaced by an adversarial training procedure by exploiting the use of a discriminator network that guides the encoder to produce latent encodings that follow a desirable prior distribution, in our case, a Gaussian.

4.2 Wasserstein Adversarial Autoencoder

As addressed in Section 2.1.3, WGANs use a method of weight clipping to impose the Lipschitz restriction in the critic network. However, this technique leads to a problematic optimization process due to the way the weight constraint and the loss function interact with each other, resulting in either disappearing or collapsing gradients without cautious adjustment of the clipping parameters c. In this work, we utilize an alternative method to enforce the Lipschitz restriction to our Adversarial Autoencoders, named Gradient penalty, as introduced by Gulrajani et al. in 2017 [17]. We introduce a new family of autoencoders, the Wasserstein Adversarial Autoencoders (WAAE).

A differentiable function is 1-Lipschtiz, if the norm of its gradients is upper bounded to 1. Thus, the gradient norm of the critic's output can be restricted by penalizing the network, when the gradient norm exceeds the target norm value of 1, rather than applying weight clipping. The objective function now becomes:

$$L = E_{\tilde{x}\sim P_g}[D(\tilde{x})] - E_{x\sim P_r}[D(x)] + \lambda \; E_{\hat{x}\sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \tag{4.4}$$

where the first two terms refer to the initial critic loss, whereas the final term refers to the gradient penalty; λ is set to 10 similar to [17], $\hat{x}$ is sampled from $\tilde{x}$ and x with t uniformly sampled between 0 and 1 in the following way:

$$\hat{x} = t \cdot \tilde{x} + (1 - t) \cdot x, \quad \text{with } 0 \leq t \leq 1 \tag{4.5}$$

To compute the gradient norm, we utilize $\hat{x}$, which is any point sampled between P_g and P_r, which is the distribution of the encoder's output and the prior distribution respectively.

In this work, WAAEs are formed similarly to the AAEs, with a few variations:

- the cost function during the regularization phase is the Wasserstein loss function, instead of the cross-entropy.
- A *linear* activation function is used in the critic's final layer to predict the score of realness for a given sample instead of predicting the likelihood of a given sample being real, which is what the *sigmoid* activation function does in the AAE.
- The training of the critic network is performed for five iterations per each iteration of the encoder.

4.3 Denoising Adversarial Autoencoder

In a Statistical Dialogue System, noise may be present as a result of the recognition and semantic errors, which can be caused by unfinished sentences, acoustic confusability or the uncertainty of natural language, along with the ambiguity of user's intention, resulting in poor performance. To alleviate this problem, we propose the use of the Denoising Adversarial Autoencoder (DAAE).

Initially, the Denoising Autoencoder [7] was used to reduce the risk of overfitting in the autoencoder [32] and remove the noise from depraved images [50]. In such situation, the input of the autoencoder was lower-dimensional than its bottleneck layer. Nevertheless, here, we maintain the model's symmetry as described in Section 4.1, where the latent encoding is of a lower dimension than the input, as our objective is to acquire a compressed representation. We construct the DAAE, which is a dense AAE, as explained in Section 4.1, designed to represent the noisy sumBS vectors, and provide robustness. More specifically, the input of a DAAE is a noisy representation $\widetilde{x}$ of the noise-free summary BS vector x:

$$\widetilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{n} \tag{4.6}$$

where n stands for the noise vector, generated by an unknown distribution and consists of the same dimensions as the noise-free summary BS and x is the target in the AAE's output. We can add synthetic noise to x by utilizing the SER percentage as probability P_{SER}. Specifically, we obtain the vector $\widetilde{x}$ by maintaining the correct semantic knowledge for a slot with probability $1 - P_{SER}$, while choosing randomly from all possible values in the ontology, with probability P_{SER}. It is important to mention that this synthetic degradation procedure can be applied to both real and simulated dialogues.

Collecting a sufficiently broad set of $[x, \tilde{x}]$ samples, we can optimize the DAAE to estimate the noise distribution and compensate with it by minimizing the average reconstruction loss specified by the mean squared error:

$$J(x, y) = E[\|x - y\|^2] \tag{4.7}$$

where x is the initial non-corrupted summary BS vector and y is the prediction of the autoencoder received from the decoder's output, when we feed the model with noisy summary BS vectors $\widetilde{x}$.

4.4 Concurrent Training Procedure

Similarly to [33], we concurrently train the policy manager and the Adversarial Autoencoders with batches composed of dialogue events. More specifically, we feed the AAE models with sumBS vectors to generate a representation. Then, we use this latent representation that is drawn from the output layer of the encoder and give it to the policy manager, so that it can be further optimized. This process is repeated for a fixed number of dialogue batches, which are collected while the dialogue simulation runs in PyDial, as can be seen in Algorithm 5.

Algorithm 5 Training Procedure

```
1:  for iteration in (1, NumOfDialogueBatches) do
2:      for dialogue in Dialogue_batch do
3:          Dialogue_Simulation.Begin()
4:          while !(Dialogue_Simulation.Finished()) = True do
5:              policy.save_Batch(Episode)
6:              AAE.save_Batch(Episode)
7:              if AAE.Batch_Full() = True then
8:                  AAE.train()
9:      policy.train()
10:     policy.evaluate()
```

Chapter 5

Experimental Evaluation

5.1 Setup - Framework

To evaluate the performance of our generative adversarial models, we made a series of experiments using the GP-SARSA policy algorithm [14] and the incremental variant of LSPI (Algorithm 2) [29] in the Pydial toolkit [49]. We decided to use GP-SARSA as it has been widely used in Dialogue Managers [6], while LSPI was selected considering its learning characteristics and its encouraging potential in SDS.

During our experiments, we train the policy algorithms off-line on samples, but they can be further optimized online in real-user environments. We evaluated the methods considered by performing experiments for four different levels of SER (0%, 15%, 30%, and 45%) in the subsequent three domains:

- *Cambridge Restaurants* (CR), which is the most familiar domain in the research of SDS with a 268-dimensional summary BS vector.
- *Laptops*11 ($LAP11$), which is one of the most challenging domains, particularly in noisy environments, with a 257-dimensional summary BS vector.
- *San Francisco Restaurants* (SFR), which composes a 636-dimensional summary BS vector.

We considered a different domain-specific AAE, WAAE, and DAAE model for each domain. The size of the network's input and output layer depends on the dimension of the sumBS vector to be represented, which is different across the three domains. The size of the input and output is the only major difference of the network, since we used the same hyper-parameters, characteristics, and optimization techniques for the Adversarial Autoencoders across all domains. More specifically, we applied the *Adam* optimizer and we used *tanh* as the activation function for the encoder and the decoder layers and the *relu* for the discriminator network. We then utilized the *dropout* method for the encoder with a ratio of 0.6 to avoid overfitting. The *learning rate* was configured as time-based, exponential decreasing, with an initial value of $(1e-02)$ and drop rate of $(1e-01)$. Finally, the batch size was set to 900 samples and the training was held for 50 epochs for every batch. We executed every experiment five separate times using five distinct initialization seeds, and we computed the mean value of the corresponding dialogue success results.

For the architectures of our networks, we constructed the models for AAE, WAAE, DAAE with 5 hidden layers for the encoder-decoder part as follows:

- **Encoder-Decoder**: input, h1: 200, h2: 100, ***h3 : 50***, h4: 100, h5: 200, output

where the input and output depends on the domain as already mentioned and h3 is the bottleneck layer. While the architecture of the discriminator network is:

- **Discriminator**: input: 50, h1: 100, h2: 100, output: 1

5.2 Experimental Results

5.2.1 Adversarial Autoencoders with various priors

During our experiments, we were interested in exploring the effect of the prior distribution we impose on our Adversarial Autoencoders on the performance of the Policy Manager. For this reason, we conducted several experiments using the AAE and WAAE, which follow two different Gaussian distributions. The first one has zero mean and unit variance, $N(0, 1)$, and the second one has zero mean and variance of five, $N(0, 5)$. We will denote the model following $N(0, 5)$ as AAE5 and WAAE5.

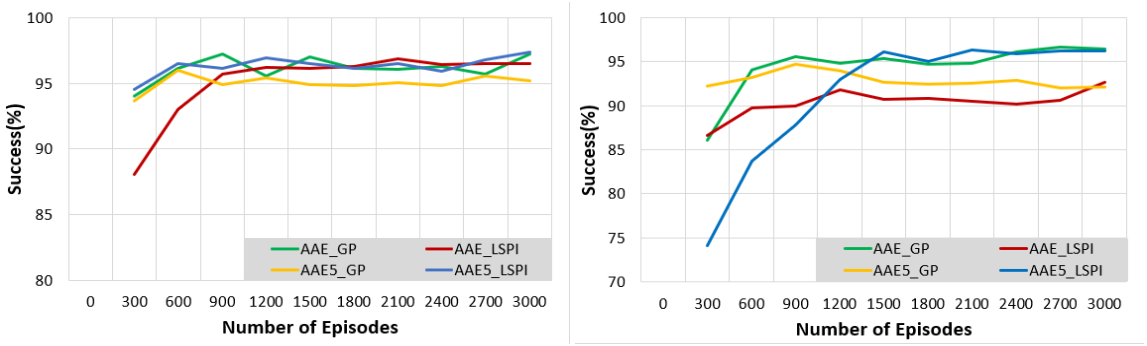
In Table 5.1, we summarize the average dialogue success in three separate domains (CR, LAP11, SFR) and for four distinct levels of noise as achieved by the AAE and AAE5 combined with GP-SARSA and LSPI policy algorithm. The results of each of the experiments were received after running for 10 batches of 300 dialogues each, i.e. 3000 dialogues in total. We notice that the AAE combined with the LSPI policy delivers the best results in the two challenging domains, LAP11 and SFR, with the most significant improvement of dialogue success occurring when the noise is greater than 15%. However, in the CR domain, which is the least demanding and complex domain, all the representations achieve success rates above 90%, even in the presence of high SER.

It is remarkable to mention that the AAE with distribution $N(0, 1)$ outperforms the AAE5 with a distribution of $N(0, 5)$, independently of the selection we make for the Policy Manager. This is mainly due to the type of information of the sumBS vector, containing values in the range of (0,1), which is closer to the normal distribution of unit variance than the one with a variance of five. The above corollary confirms the reason behind the high performance of the Variational Autoencoder used in [33], which produces latent encodings that follow a normal distribution of zero mean and unit variance. Nevertheless, we observe that, even when we use AAE5, we maintain a great performance above 80% in the two difficult domains.

		Domains		
SER	BS	CR	LAP11	SFR
0%	AAE-GP	97.2%(±2.2)	94.2%(±2.6)	96.0%(±2.2)
	AAE5-GP	95.2%(±2.4)	94.6%(±2.6)	94.8%(±2.4)
	AAE-LSPI	96.5%(±1.8)	**98.2%(±1.5)**	**97.4%(±1.7)**
	AAE5-LSPI	**97.4%(±1.3)**	93.1%(±2.7)	96.5%(±1.8)
15%	AAE-GP	**96.4%(±2.1)**	92.1% (±3.0)	93.7%(±2.7)
	AAE5-GP	92.1%(±3.0)	91.5%(±3.1)	92.0%(±3.0)
	AAE-LSPI	92.6%(±2.8)	**97.7%(±1.5)**	**97.2%(±1.8)**
	AAE5-LSPI	96.2%(±2.0)	90.3%(±3.3)	90.2%(±3.2)
30%	AAE-GP	**96.2%(±1.8)**	89.4%(±3.4)	89.8%(±3.4)
	AAE5-GP	91.2%(±3.1)	88.8%(±3.7)	88.0%(±3.6)
	AAE-LSPI	92.0%(±2.9)	**96.8%(±1.5)**	**94.9%(±2.4)**
	AAE5-LSPI	93.7%(±2.4)	89.2%(±3.4)	82.2%(±4.3)
45%	AAE-GP	**94.5%(±2.0)**	87.7%(±3.8)	88.1%(±3.6)
	AAE5-GP	91.3%(±3.1)	86.7%(±3.9)	83.1%(±4.2)
	AAE-LSPI	91.1%(±3.0)	**93.4%(±1.9)**	**91.9%(±2.8)**
	AAE5-LSPI	92.8%(±2.5)	84.0%(±4.0)	76.3%(±4.8)

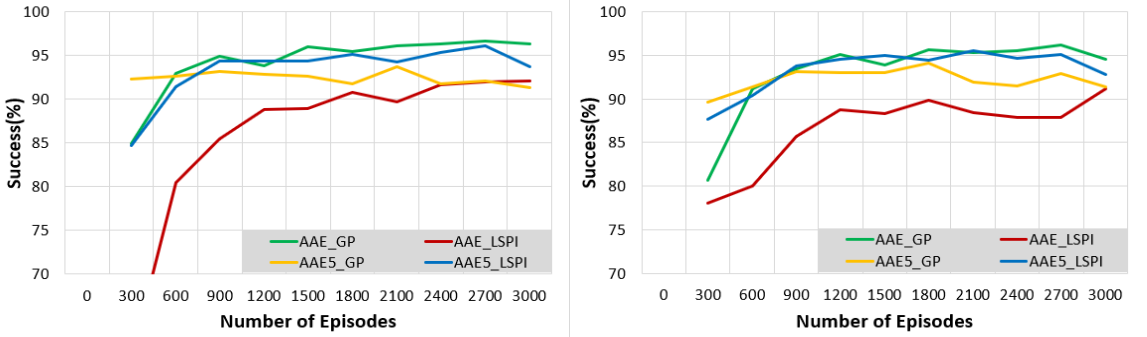
Table 5.1: Mean dialogue success while applying AAE and AAE5 which follow $N(0,1)$ and $N(0,5)$ distibutions respectively. Best result in bold. The parenthesis depicts the standard deviation.

The diagrams in Figures 5.1, 5.2 and 5.3, show the progression of the average dialogue success during optimization in the three domains for four distinct levels of noise. We notice that LSPI, in combination with our models, demands more training samples in order to converge compared to GP-SARSA. This phenomenon is more intense in the case of AAE5, where more than 2000 dialogue samples are needed, while all other combinations of the PM with AAE require roughly 1500 samples. In addition, the optimization of LSPI initiates with much lower success rates compared to GP-SARSA, especially when AAE5 is in use in LAP11 and SFR domains. Even though LSPI eventually achieves the highest dialogue success rates, GP-SARSA provides a more stable training procedure and a quicker convergence.



(a) CR 0%

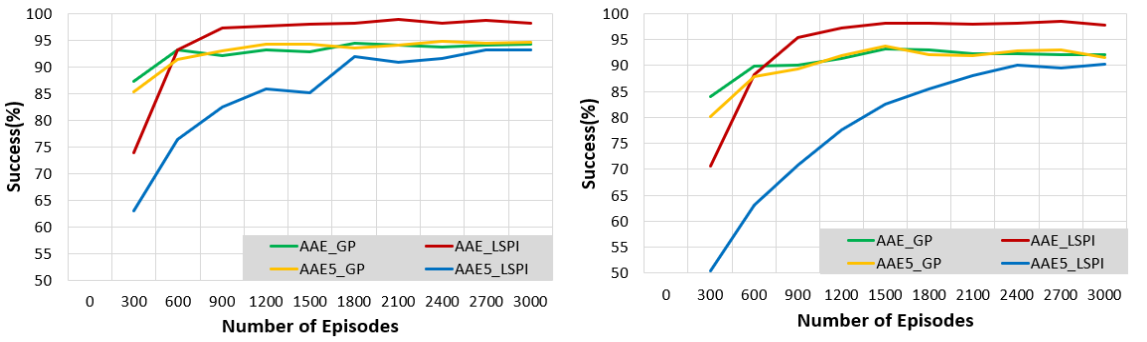
(b) CR 15%



(c) CR 30%

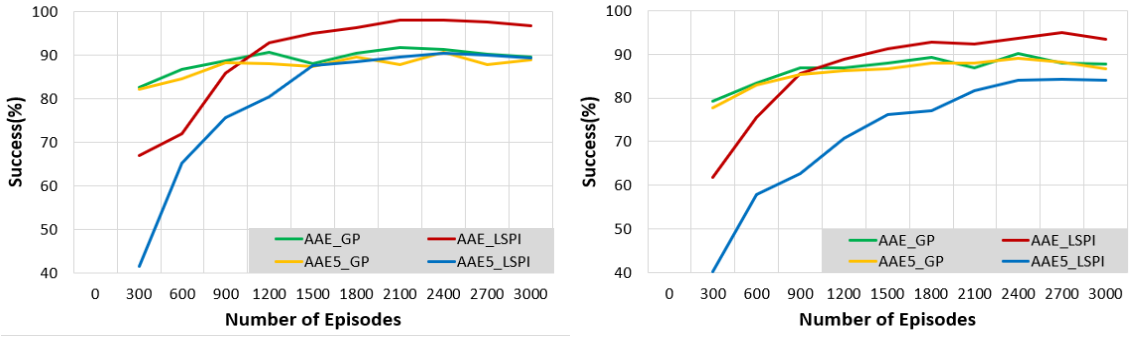
(d) CR 45%

Figure 5.1: Average Success in CR domain for different noise levels and state representations with GP-SARSA and LSPI.



(a) LAP11 0%

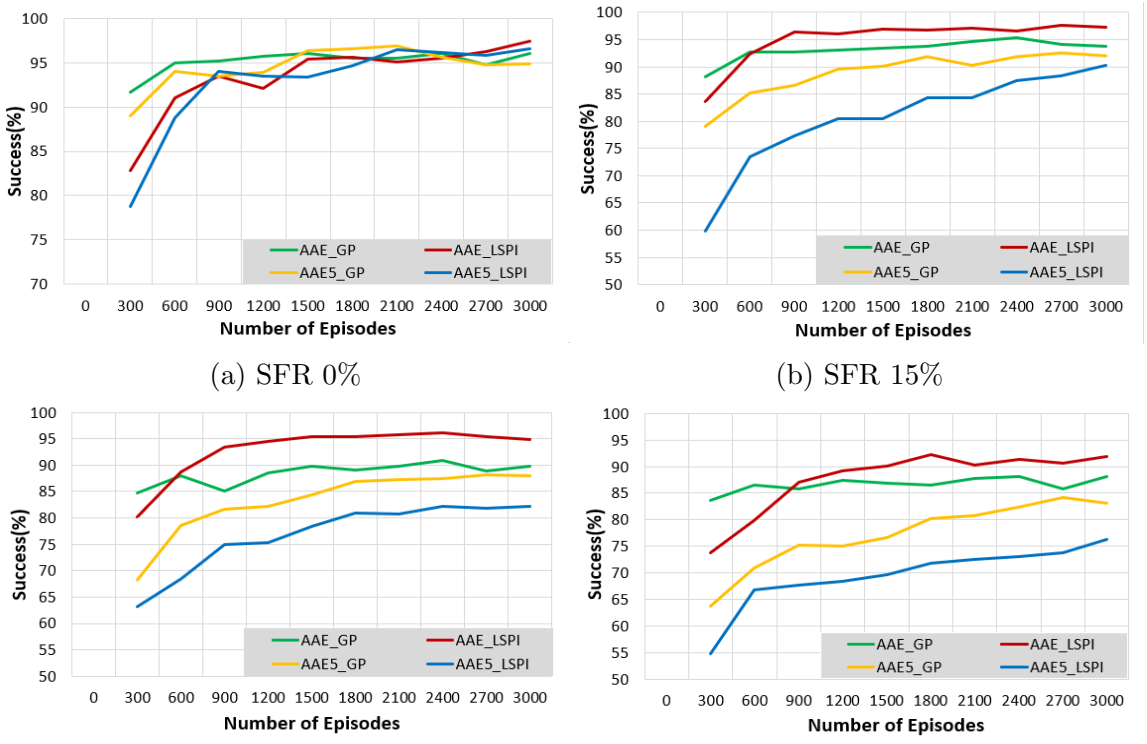
(b) LAP11 15%



(c) LAP11 30%

(d) LAP11 45%

Figure 5.2: Average Success in LAP11 domain for different noise levels and state representations with GP-SARSA and LSPI.



(a) SFR 0%

(b) SFR 15%

(c) SFR 30%

(d) SFR 45%

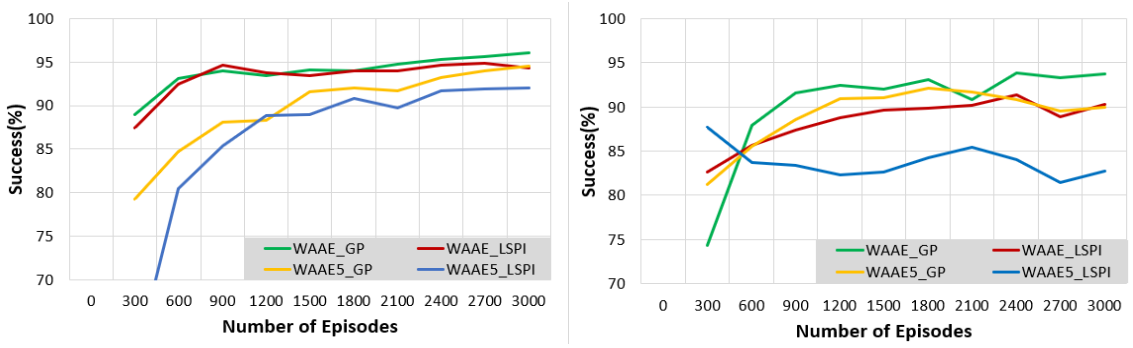
Figure 5.3: Average Success in SFR domain for different noise levels applying and state representations with GP-SARSA and LSPI.

In Table 5.2, we illustrate the average dialogue success as achieved by the WAAE and WAAE5 combined with GP-SARSA and LSPI policy algorithms. The degradation of the performance is again confirmed, when we impose a normal distribution with a variance of 5 to our Adversarial Autoencoders and, more specifically, to the WAAE. We observe that the WAAE5 representations deliver lower dialogue success rates than WAAE, independently of the Policy Manager we select. To be more precise, we discern a dialogue success reduction of nearly up to 30% in the LAP11 domain and 15% in the SFR domain. It is worth noting that when we apply WAAE, GP-SARSA presents the best performance across all domains and noise levels. Finally, if we compare Tables 5.1 and 5.2, it becomes evident that the AAE outperforms the WAAE in all domains and environments with a significant difference appearing when SER is greater than 15%.

		Domains		
SER	BS	CR	LAP11	SFR
0%	WAAE-GP	**96.0%(±2.1)**	**90.5%(±3.2)**	90.1%(±3.5)
	WAAE5-GP	94.5%(±2.5)	66.9%(±5.3)	79.3%(±4.4)
	WAAE-LSPI	94.3%(±2.6)	80.1%(±4.8)	**91.2%(±2.8)**
	WAAE5-LSPI	95.8%(±2.0)	51.0%(±5.6)	74.4%(±4.8)
15%	WAAE-GP	**93.7%(±2.7)**	**81.8%(±4.2)**	**76.3%(±4.6)**
	WAAE5-GP	89.9%(±3.2)	58.0%(±5.5)	72.4%(±5.0)
	WAAE-LSPI	90.3%(±3.3)	58.7%(±5.4)	67.6%(±5.5)
	WAAE5-LSPI	82.7%(±4.0)	43.8%(±5.2)	61.6%(±5.4)
30%	WAAE-GP	**93.5%(±2.7)**	**76.2%(±4.7)**	**75.2%(±4.7)**
	WAAE5-GP	86.6%(±3.8)	57.4%(±5.7)	69.2%(±5.2)
	WAAE-LSPI	88.4%(±3.5)	40.8%(±5.0)	63.1%(±5.3)
	WAAE5-LSPI	80.2%(±4.3)	41.7%(±5.5)	57.6%(±5.5)
45%	WAAE-GP	**87.1%(±3.7)**	**58.4%(±5.5)**	**73.0%(±5.0)**
	WAAE5-GP	84.7%(±4.0)	53.0%(±5.7)	67.2%(±5.3)
	WAAE-LSPI	86.4%(±3.8)	38.8%(±5.1)	53.2%(±5.5)
	WAAE5-LSPI	76.8%(±4.6)	36.3%(±5.4)	51.2%(±5.6)

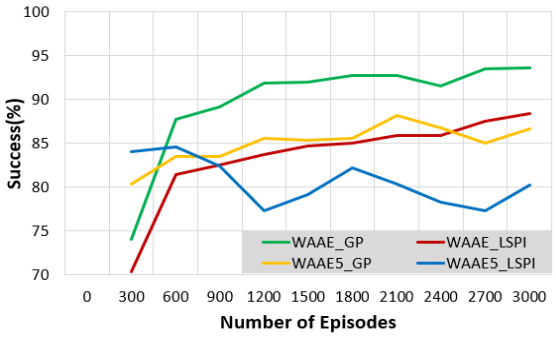
Table 5.2: Mean dialogue success while applying WAAE and WAAE5 which follow $N(0,1)$ and $N(0,5)$ distibutions respectively. Best result in bold. The parenthesis depicts the standard deviation.

In Figures 5.4, 5.5 and 5.6, we display the progression of the average dialogue success during the training of the WAAE and WAAE5 in combination with GP-SARSA and LSPI in the three domains for four distinct levels of noise. We notice that all curves present significant and frequent fluctuations throughout the optimization process, especially in the two most challenging domains (LAP11 and SFR). Moreover, we discern a drop in the system's performance after 1500 dialogue samples, which leads to unstable training and weakness to converge. This behavior is mainly due to the low discriminator loss and high generator loss as noticed during optimization, meaning that the discriminator does a good job at distinguishing the real prior distribution from the generated samples, making it harder for the encoder to match the desirable prior distribution and leading to poor performance of the Autoencoder. This does not support our initial hypothesis that the WAAE would provide more stable training and better matching of the desirable prior distribution and eventually deliver better performance than the AAE.

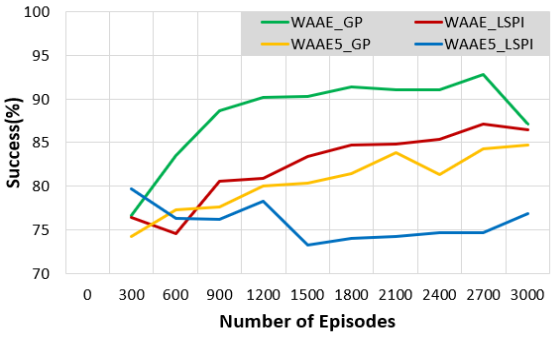


(a) CR 0%

(b) CR 15%

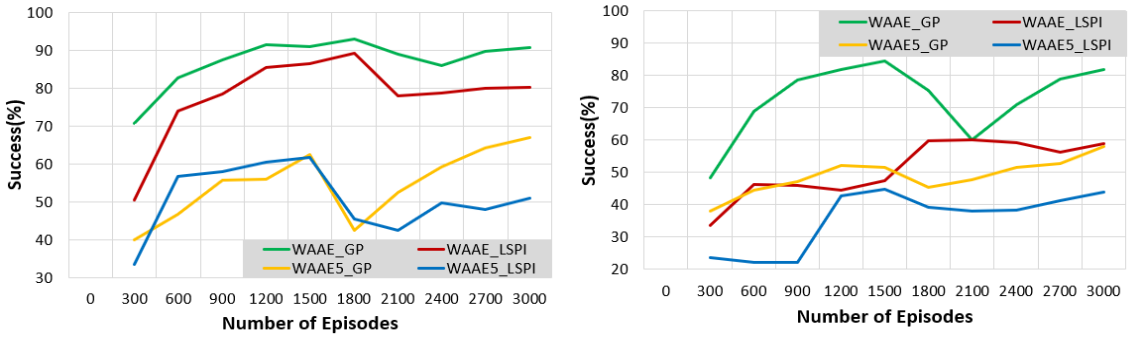


(c) CR 30%



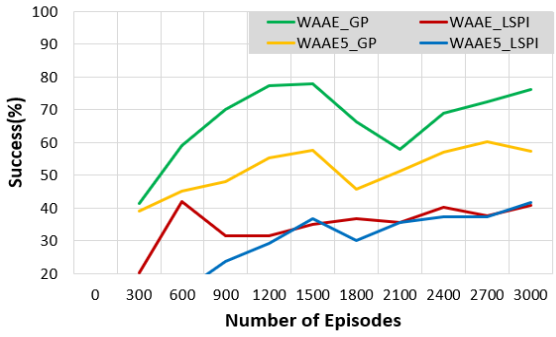
(d) CR 45%

Figure 5.4: Average Success in CR domain for different noise levels and state representations with GP-SARSA and LSPI.

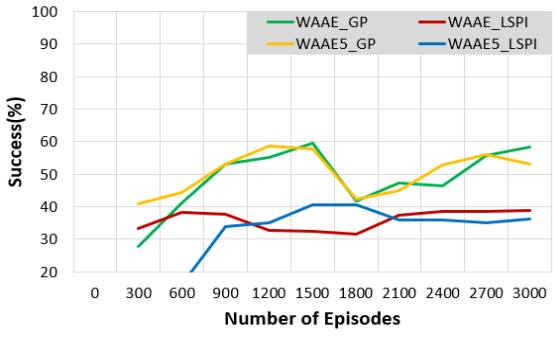


(a) LAP11 0%

(b) LAP11 15%

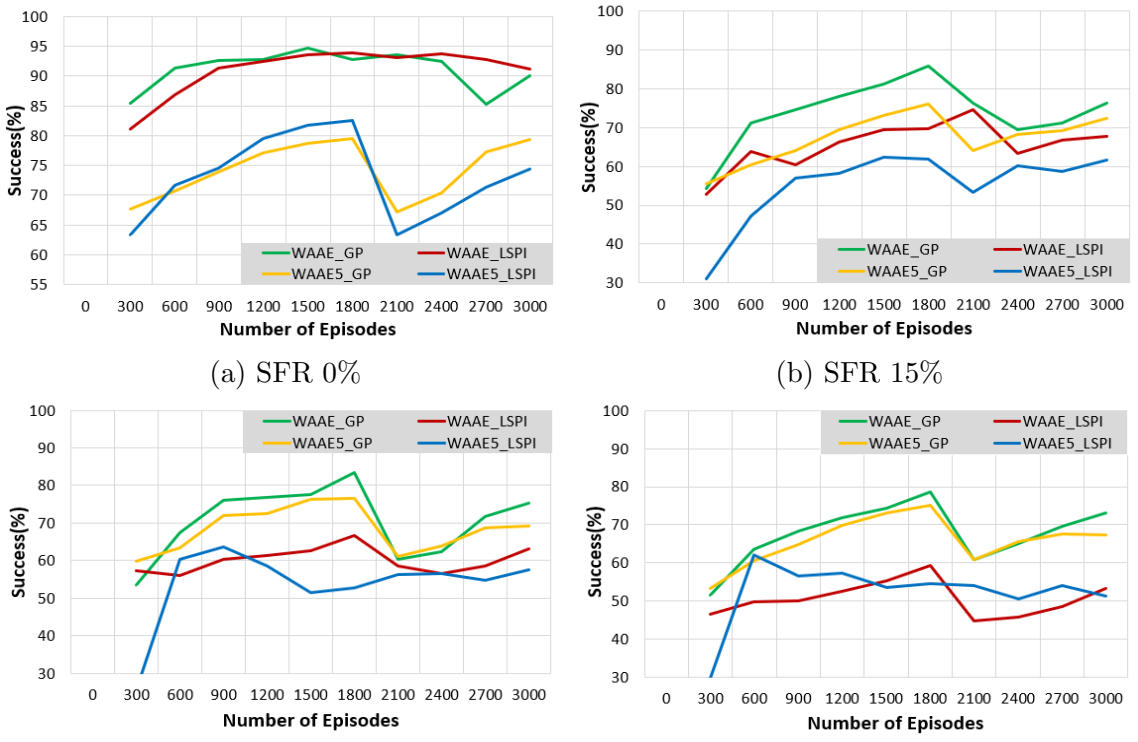


(c) LAP11 30%



(d) LAP11 45%

Figure 5.5: Average Success in LAP11 domain for different noise levels and state representations with GP-SARSA and LSPI.



(a) SFR 0%

(b) SFR 15%

(c) SFR 30%

(d) SFR 45%

Figure 5.6: Average Success in SFR domain for different noise levels and state representations with GP-SARSA and LSPI.

5.2.2 Vanilla vs Wasserstein Adversarial Autoencoder

In Table 5.3, we show the average success of the dialogue in three separate domains (CR, LAP11, SFR) and for four distinct levels of noise, when we utilize the representations received from the AAE and WAAE that follow the normal distribution $N(0, 1)$ with only the GP-SARSA policy algorithm. We compare them to the success rates using the sumBS vector and the AE representations [33].

We show that the AAE representation obtains the best performance among all the domains and environments compared to the other models except for the CR domain with 0% and 15% of noise, where the sumBS and the AE seem to produce slightly better scores. Even though performance degradation exists, especially in the two most challenging domains (LAP11 and SFR), we observe that as the noise level increases, the AAE achieves considerably higher dialogue success rates than the sumBS and AE representations. More specifically, when we apply AAE, GP-SARSA is able to obtain success rates close to 90% in LAP11 and SFR domains with 30% and 45% of SER, compared to 50% and 70% in the LAP11 and SFR domains, respectively when AE or sumBS are used.

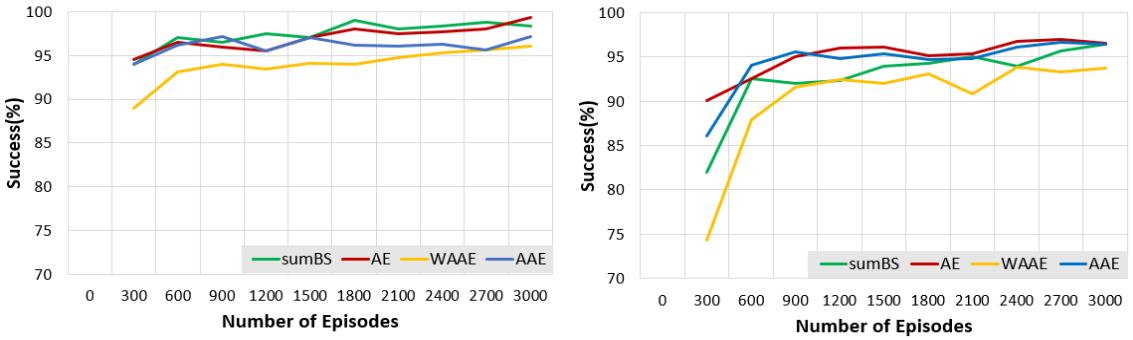
Moreover, we notice that when we utilize WAAE, GP-SARSA cannot reach the performance achieved when applying AAE. The significant difference in dialogue success rates appears in LAP11 and SFR domains with high SER, where AAE clearly outperforms WAAE. This reveals that the modified version of AAE does not benefit

		Domains		
SER	BS	CR	LAP11	SFR
0%	sumBS	98.4%(±1.1)	86.8%(±3.8)	95.2%(±1.3)
	AE	**99.3%(±0.5)**	92.6%(±1.9)	95.3%(±0.8)
	AAE	97.2%(±2.2)	**94.2%(±2.6)**	**96.0%(±2.2)**
	WAAE	96.0%(±2.1)	90.5%(±3.2)	90.1%(±3.5)
15%	sumBS	96.4%(±2.2)	66.5%(±2.3)	81.6%(±1.6)
	AE	**96.5%(±1.0)**	68.9%(±10.1)	89.3%(±1.8)
	AAE	96.4%(±2.1)	**92.1%(±3.0)**	**93.7%(±2.7)**
	WAAE	93.7%(±2.7)	81.8%(±4.2)	76.3%(±4.6)
30%	sumBS	88.5%(±4.2)	51.4%(±9.3)	66.3%(±5.3)
	AE	92.2%(±1.1)	50.1%(±10.4)	69.4%(±2.3)
	AAE	**96.2%(±1.8)**	**89.4%(±3.4)**	**89.8%(±3.4)**
	WAAE	93.5%(±2.7)	76.2%(±4.7)	75.2%(±4.7)
45%	sumBS	78.0%(±3.4)	24.1%(±5.5)	53.9%(±6.8)
	AE	78.1%(±3.3)	38.7%(±5.4)	36.9%(±7.9)
	AAE	**94.5%(±2.0)**	**87.7%(±3.8)**	**88.1%(±3.6)**
	WAAE	87.1%(±3.7)	58.4%(±5.5)	73.0%(±5.0)

Table 5.3: Mean dialogue success with GP-SARSA for different state representations. Best result in bold. The parenthesis depicts the standard deviation.

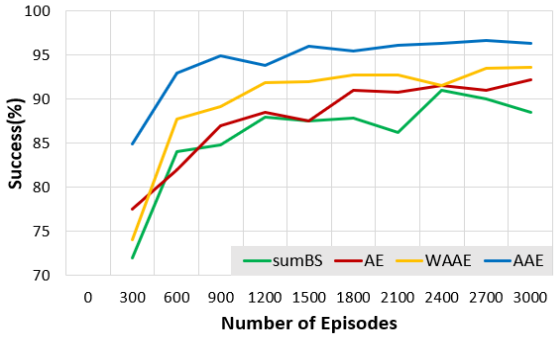
the PM to obtain better results. This is mainly due to the low discriminator loss and high generator loss, as noticed during optimization, meaning that the discriminator does a good job at distinguishing the real prior distribution from the generated samples, making it harder for the encoder to match the desirable prior distribution and leading to poor performance of the Autoencoder. However, GP-SARSA combined with WAAE exhibits better performance than the AE and SumBS representations when the noise is 30% or greater, which once again confirms the efficiency of our adversarial models.

In Figures 5.7, 5.8, and 5.9, we demonstrate the progress of the average dialogue success achieved by the representations mentioned above in the three domains for four distinct levels of noise. We observe that GP-SARSA requires approximately 1500 dialogues to reach convergence when AAE is applied, while in the case of sumBS and AE it takes roughly 2000 dialogues. Furthermore, we notice that WAAE does not help the PM to converge and achieve similar performance to AAE, but it presents several fluctuations throughout its training, especially in the two difficult domains (LAP11 and SFR).

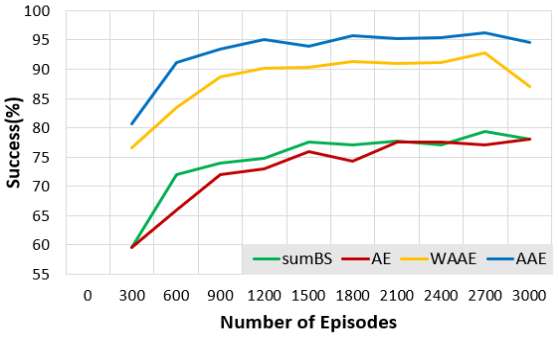


(a) CR 0%

(b) CR 15%

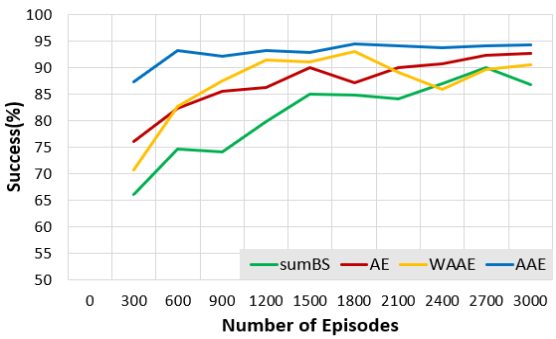


(c) CR 30%

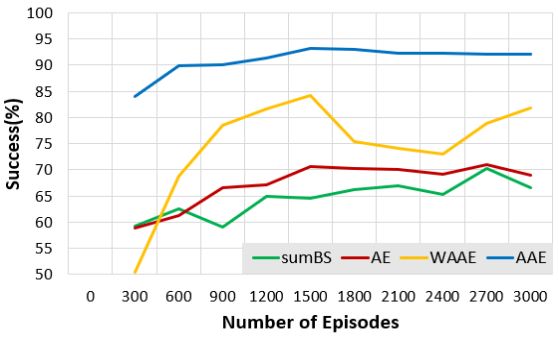


(d) CR 45%

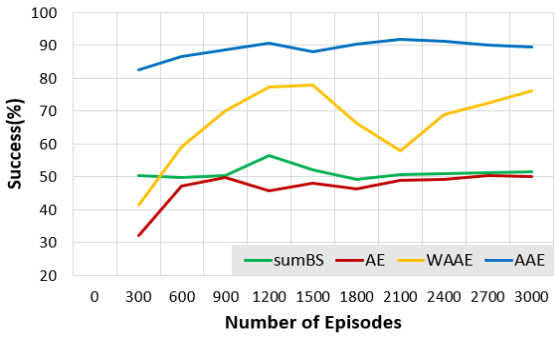
Figure 5.7: Average Success in CR domain for different noise levels and state representations with GP-SARSA.



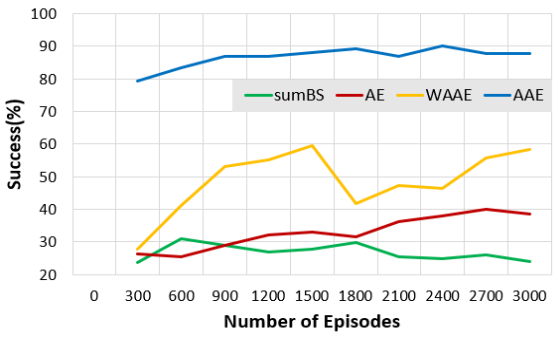
(a) LAP11 0%



(b) LAP11 15%

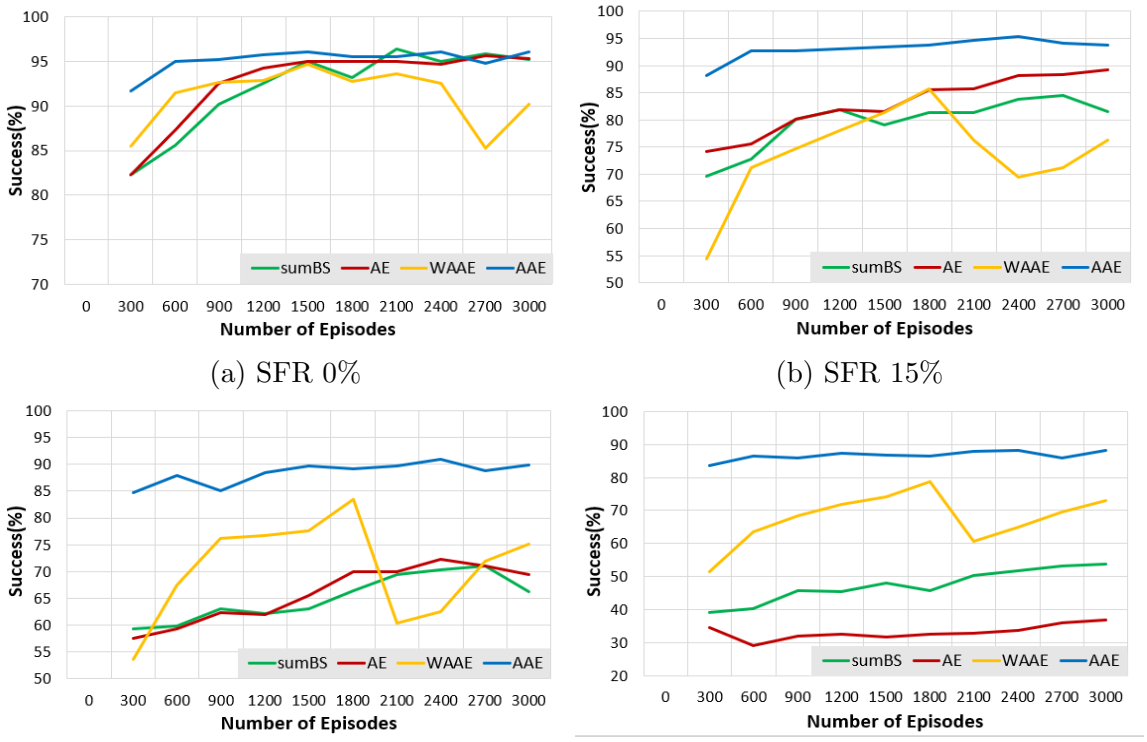


(c) LAP11 30%



(d) LAP11 45%

Figure 5.8: Average Success in LAP11 domain for different noise and state representations with GP-SARSA.



(a) SFR 0%

(b) SFR 15%

(c) SFR 30%

(d) SFR 45%

Figure 5.9: Average Success in SFR domain for different noise levels and state representations with GP-SARSA.

5.2.3 Denoising Adversarial Autoencoders

In the final part of our experiments, we compare the performance of our Adversarial Autoencoders with the performance of the Variational Denoising Autoencoder (VDAE), which delivered the best results in [33]. So far, the AAE has proved to achieve quite better performance than the WAAE. Thus, in the following experiments, we utilize only the AAE which follows a Gaussian distribution with unit variance as it proved to deliver better results that the one with variance of five. As we want to have a fair comparison, we also use the denoising mechanism for the AAE as described in Chapter 4, and we call it DAAE.

In Table 5.4, we present the average dialogue success achieved by the AAE and DAAE representations combined with both policy algorithms, GP-SARSA, and LSPI in the three environments where noise is present. We notice that the DAAE offers a slight improvement in the CR domain and a bigger one in the LAP11 and SFR domains with 45% SER. Nevertheless, the AAE exhibits very high performance, quite close to that of the DAAE, even in the two challenging domains with increased noise. Finally, we discern that LSPI combined with the AAE or DAAE provide better results than GP-SARSA in most environments.

		Domains		
SER	BS	CR	LAP11	SFR
15%	AAE-GP	96.4%(±2.1)	92.1% (±3.0)	93.7%(±2.7)
	AAE-LSPI	92.6%(±2.8)	**97.7%(±1.5)**	97.2%(±1.8)
	DAAE-GP	96.6%(±2.1)	92.6%(±2.9)	93.0%(±2.8)
	DAAE-LSPI	**97.1%(±1.6)**	97.4%(±1.7)	**97.4%(±1.8)**
30%	AAE-GP	**96.2%(±1.8)**	89.4%(±3.4)	89.8%(±3.4)
	AAE-LSPI	92.0%(±2.9)	96.8%(±1.5)	94.9%(±2.4)
	DAAE-GP	96.0%(±2.3)	91.3%(±3.2)	91.2%(±3.1)
	DAAE-LSPI	93.6%(±1.8)	**97.1%(±1.7)**	**95.9%(±2.2)**
45%	AAE-GP	94.5%(±2.0)	87.7%(±3.8)	88.1%(±3.6)
	AAE-LSPI	91.1%(±3.0)	93.4%(±1.9)	91.9%(±2.8)
	DAAE-GP	**94.9%(±2.4)**	89.1%(±3.5)	89.3%(±3.5)
	DAAE-LSPI	92.6%(±2.8)	**96.2%(±2.0)**	**94.8%(±2.4)**

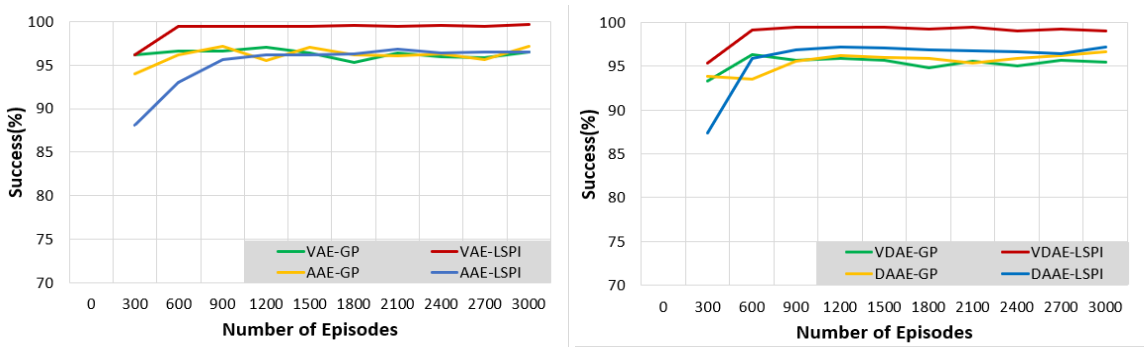
Table 5.4: Mean dialogue success while applying AAE and DAAE. Best result in bold. The parenthesis depicts the standard deviation.

In Table 5.5, we compare the dialogue success rates produced by the DAAE and the ones provided by the VDAE, which exhibited the best performance among all other AE-based representations in [33]. When the SER is 0%, the DAAE is identical to AAE, and VDAE is equal to VAE; thus, we only include the AAE and VAE success rates. We observe that the DAAE combined with GP-SARSA outperforms the combination of VDAE with GP-SARSA in almost every environment and domain. However, DAAE combined with LSPI exhibits lower performance than VDAE combined with LSPI, especially in the CR domain. In the LAP11 domain, the performance of the DAAE-LSPI combination is very close to that of VDAE-LSPI, where there is a difference of less than 1% of dialogue success rate. Finally, the DAAE outperforms VDAE in the SFR domain regardless of the selection of the Policy Manager. Despite the high performance of the DAAE, which is very close to that of the VDAE, the DAAE produces higher standard deviation, especially in the two difficult domains of LAP11 and SFR.

		Domains		
SER	BS	CR	LAP11	SFR
0%	AAE-GP	97.2%(±2.2)	94.2%(±2.6)	96.0%(±2.2)
	AAE-LSPI	96.5%(±1.8)	98.2%(±1.5)	97.4%(±1.7)
	VAE-GP	96.5%(±2.7)	94.7%(±1.7)	93.7%(±2.1)
	VAE-LSPI	**99.7%(±0.4)**	**98.5%(±1.1)**	**98.4%(±1.1)**
15%	DAAE-GP	96.6%(±2.1)	92.6%(±2.9)	93.0%(±2.8)
	DAAE-LSPI	97.1%(±1.6)	97.4%(±1.7)	**97.4%(±1.8)**
	VDAE-GP	95.5%(±0.8)	91.7%(±0.5)	93.6%(±1.0)
	VDAE-LSPI	**99.0%(±0.8)**	**97.9%(±0.6)**	97.0%(±0.7)
30%	DAAE-GP	96.0%(±2.3)	91.3%(±3.2)	91.2%(±3.1)
	DAAE-LSPI	93.6%(±1.8)	97.1%(±1.7)	**95.9%(±2.2)**
	VDAE-GP	92.9%(±1.3)	90.2%(±1.9)	89.9%(±2.7)
	VDAE-LSPI	**98.1%(±1.2)**	**97.7%(±0.9)**	95.3%(±1.6)
45%	DAAE-GP	**94.9%(±2.4)**	89.1%(±3.5)	89.3%(±3.5)
	DAAE-LSPI	92.6%(±2.8)	96.2%(±2.0)	**94.8%(±2.4)**
	VDAE-GP	92.3%(±3.3)	87.9%(±2.7)	88.6%(±2.9)
	VDAE-LSPI	93.6%(±2.7)	**96.7%(±0.9)**	94.6%(±1.5)

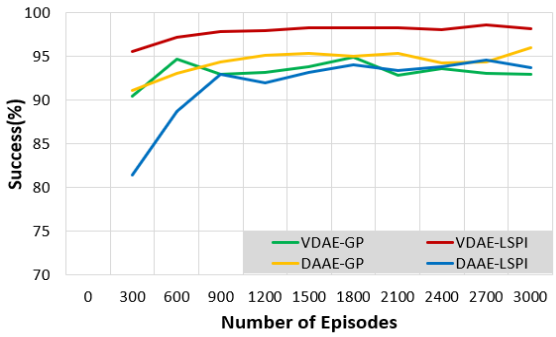
Table 5.5: Mean dialogue success while applying DAAE versus VDAE. Best result in bold. The parenthesis depicts the standard deviation.

In Figures 5.10, 5.11, and 5.12, we present the progression of the average dialogue success accomplished by the DAAE and VDAE representations in the three domains and for four distinct levels of noise. We observe that the LSPI policy algorithm requires approximately 1200 dialogue samples to converge regardless of the selection of the Autoencoder, while GP-SARSA needs somewhere between 300 and 600 dialogue samples. In the CR domain, the DAAE combined with LSPI requires by far the most training samples to reach convergence, especially in the 45% noise environment, where it takes more than 2000 samples. Moreover, the DAAE initiates the optimization process from lower dialogue success rates than the VDAE independent of the Policy Manager we utilize. This is mainly due to the Adversarial Autoencoders' hard training procedure, where it takes more time for the encoder to learn and produce the desirable distribution compared to the Variational Autoencoder. However, the DAAE is finally able to reach the performance of the VDAE.

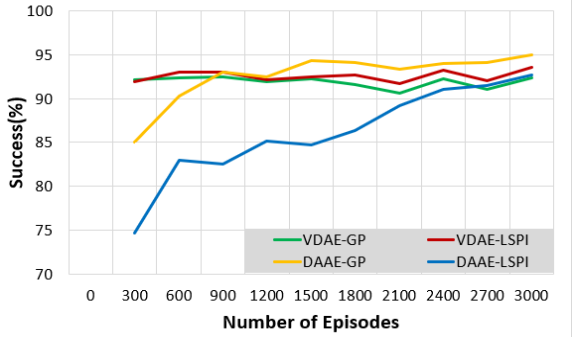


(a) CR 0%

(b) CR 15%

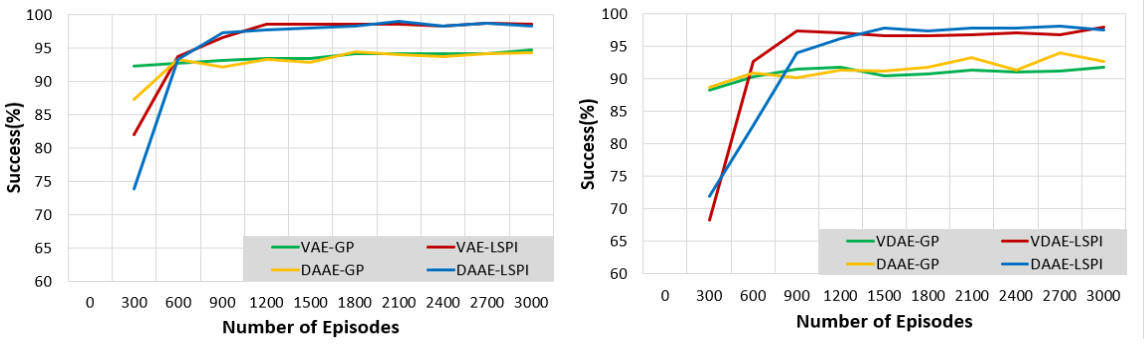


(c) CR 30%



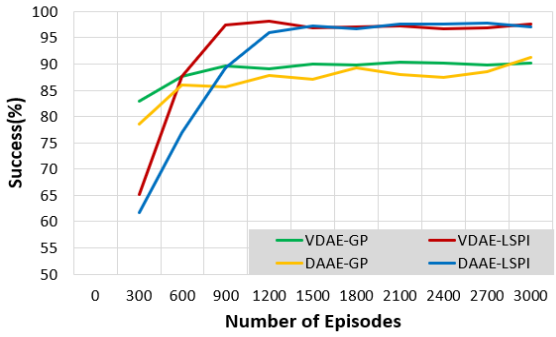
(d) CR 45%

Figure 5.10: Average Success in CR domain for different noise levels and state representations with GP-SARSA and LSPI.

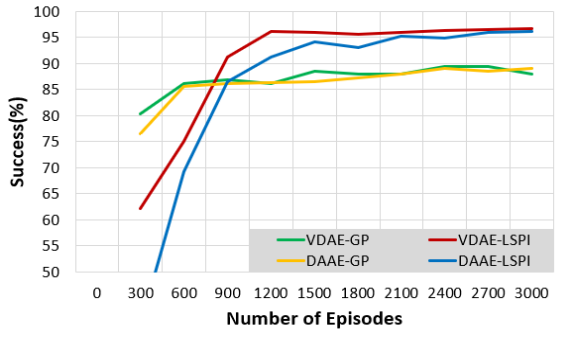


(a) LAP11 0%

(b) LAP11 15%

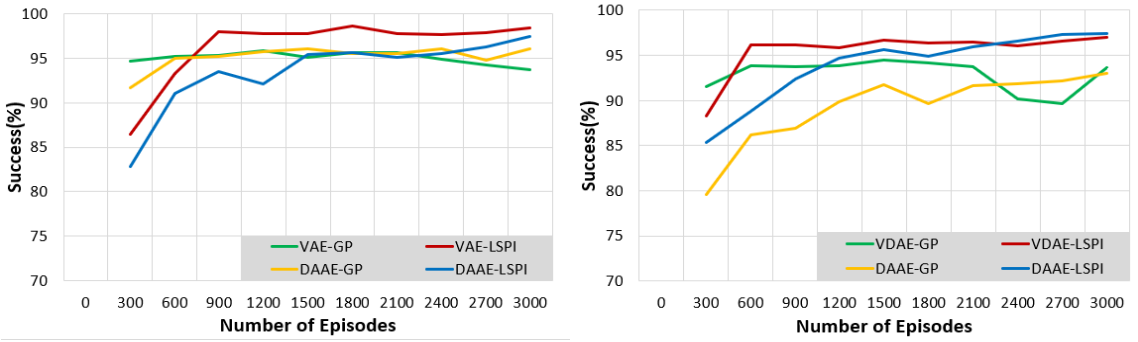


(c) LAP11 30%



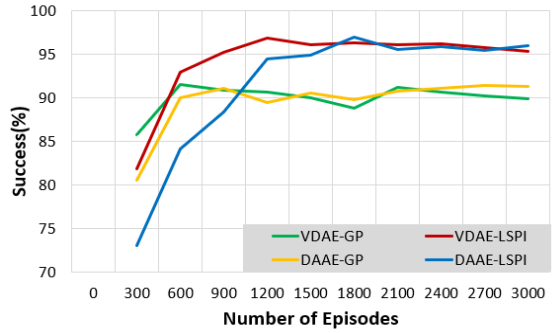
(d) LAP11 45%

Figure 5.11: Average Success in LAP11 domain for different noise levels and state representations with GP-SARSA and LSPI.

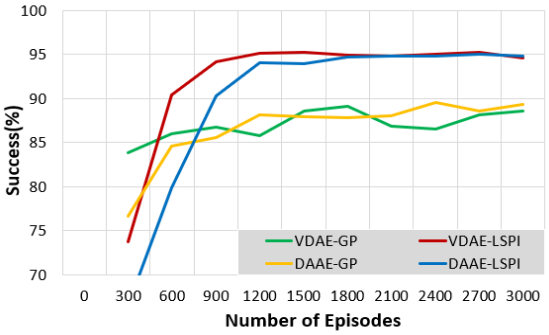


(a) SFR 0%

(b) SFR 15%



(c) SFR 30%



(d) SFR 45%

Figure 5.12: Average Success in SFR domain for different noise levels and state representations with GP-SARSA and LSPI.

Chapter 6

Conclusion & Future Work

In this work, we investigate the effectiveness of Generative Adversarial Networks (GANs) within the framework of a Spoken Dialogue System. More specifically, we introduce an innovative use of the Adversarial Autoencoder, which is trained like a GAN, in our effort to explore more efficient Belief State-Space representations that could help the Policy Manager achieve high performance.

First, we present the background of GANs and describe their architecture and their adversarial training procedure. Then, we review some of the most significant difficulties that arise when a GAN is trained, to state that this process is not as straightforward as training a standard Neural Network. Next, we discuss a technique that provides GANs with more stable training and helps overcome most of the difficulties that arise during training. More specifically, this method uses a new loss function based on the Wasserstein distance to optimize the network's parameters, forming a new family of GANs, called Wasserstein Generative Adversarial Networks (WGANs).

Next, we describe our proposed method to represent the Belief State (BS), which is based on the AAE, and we compare it with the Variational Autoencoder, which was used in [33]. We then introduce the Wasserstein Adversarial Autoencoder, which has many features in common with WGANs, as we want to explore the efficiency of such a technique in Adversarial Autoencoders. We conclude the chapter of our implementation with the Denoising Adversarial Autoencoder, which is trained to provide meaningful BS representations in environments with high noise levels and our concurrent training procedure, which is quite similar to the one used in [33] and allows the Adversarial models to be optimized in parallel with the Policy Manager.

Our experiments were conducted in the Pydial toolkit, where we utilized two RL policy algorithms, the GP-SARSA and the LSPI. We tested our models in three domains (CR, LAP11, and SFR) for four distinct levels of noise (0%, 15%, 30%, 45%), and we compared the performance of our models with the current state-of-the-art [34, 8]. We first experimented on the impact of the prior distribution on the performance of the Adversarial Autoencoders and, as a result, in the performance of the Policy Manager. We tested our models by imposing two Gaussian distributions, both having zero mean, the one with unit variance, and the other with a variance of five. The results revealed that the normal prior distribution with unit variance helps both the AAE and the WAAE achieve higher dialogue success rates than those with a variance of five, regardless of the selection we make for the PM. Next, we

compared the performance of the AAE and the WAAE to the baseline sumBS vector and the vanilla AE. The results showed that the AAE and the WAAE achieve better performance than the baseline sumBS vector and the vanilla AE, especially in the noisiest environments. However, the AAE outperformed the WAAE in all cases and presented smoother optimization, which proved our initial hypothesis, that the Wasserstein loss function would improve the performance of the AAE, wrong.

In the final section of our experiments, we examined the efficiency of the DAAE in noisy environments. It proved to provide a slight improvement in the performance compared to the AAE, which states that the AAE is able to provide noise-robust BS representations even in the most challenging domains with high SER. Finally, it was essential to compare the performance of the DAAE to that of the VDAE, which exhibited the best success rates in [33]. We showed that the DAAE outperforms the VDAE, when we utilize GP-SARSA as our PM, and we can obtain very high dialogue success rates close to that produced by the VDAE even when LSPI is used. Nevertheless, both models present similar behavior during the training procedure, with the DAAE starting the optimization from lower success rates, but eventually reaching the performance of the VDAE.

Consequently, in this diploma thesis, we propose an innovative use of the Adversarial Autoencoder as a method to represent the summary BS space in a more compact and lower-dimensional vector. We also introduced the Wasserstein Adversarial Autoencoder, which eventually fell behind our initial expectations. Besides, we experimented on two different prior distributions and showed the impact they have on the performance of the Policy Manager. Finally, we utilized a denoising mechanism for the AAE, which proved to provide only a slight improvement in the performance, showing the ability of the AAE to produce high-quality sumBS representations even in very noisy environments.

Future work concerns further experimentation on different prior distributions apart from the Gaussian. Also, the representations obtained by the AAE could be used by non-linear Policy Managers, which is another family of RL algorithms that were not discussed in this thesis, but offer great potential. Finally, the research could focus on developing a policy algorithm based on Generative Adversarial Networks in order to explore another aspect of this popular and highly potential category of Neural Networks.

References

[1] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. "Face aging with conditional generative adversarial networks". In: *2017 IEEE international conference on image processing (ICIP)*. IEEE. 2017, pp. 2089–2093.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN". In: *arXiv preprint arXiv:1701.07875* (2017).

[3] Richard Bellman. "A Markovian decision process". In: *Journal of mathematics and mechanics* (1957), pp. 679–684.

[4] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. "Neural photo editing with introspective adversarial networks". In: *arXiv preprint arXiv:1609.07093* (2016).

[5] Paweł Budzianowski, Stefan Ultes, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Inigo Casanueva, Lina Rojas-Barahona, and Milica Gašić. "Sub-domain modelling for dialogue management with hierarchical reinforcement learning". In: *arXiv preprint arXiv:1706.06210* (2017).

[6] Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Stefan Ultes, Lina Rojas-Barahona, Steve Young, and Milica Gašić. "A benchmarking environment for reinforcement learning based task oriented dialogue management". In: *arXiv preprint arXiv:1711.11023* (2017).

[7] Antonia Creswell and Anil Anthony Bharath. "Denoising adversarial autoencoders". In: *IEEE transactions on neural networks and learning systems* 30.4 (2018), pp. 968–984.

[8] Vassilios Diakoloukas, Fotios Lygerakis, Michail G Lagoudakis, and Margarita Kotti. "Variational Denoising Autoencoders and Least-Squares Policy Iteration for Statistical Dialogue Managers". In: *IEEE Signal Processing Letters* (2020).

[9] P Kingma Diederik and Max Welling. "Auto-encoding variational bayes". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Vol. 1. 2014.

[10] David A Edwards. "On the Kantorovich–Rubinstein theorem". In: *Expositiones Mathematicae* 29.4 (2011), pp. 387–398.

[11] Arthur Szlam Emily L. Denton Soumith Chintala and Rob Fergus. "Deep generative image models using a laplacian pyramid of adversarial networks". In: *Advances in neural information processing systems*. 2015, pp. 1486–1494.

[12] Mehdi Fatemi, Layla El Asri, Hannes Schulz, Jing He, and Kaheer Suleman. "Policy networks with two-stage training for dialogue systems". In: *arXiv preprint arXiv:1606.03152* (2016).

[13] Milica Gašić, Filip Jurčiček, Blaise Thomson, Kai Yu, and Steve Young. "On-line policy optimisation of spoken dialogue systems via live interaction with human subjects". In: *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE. 2011, pp. 312–317.

[14] Milica Gašić and Steve Young. "Gaussian processes for pomdp-based dialogue manager optimization". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.1 (2013), pp. 28–40.

[15] Ian J Goodfellow. "On distinguishability criteria for estimating generative models". In: *arXiv preprint arXiv:1412.6515* (2014).

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. "Improved training of Wasserstein GANs". In: *Advances in neural information processing systems*. 2017, pp. 5767–5777.

[18] Hilda Hardy, Tomek Strzalkowski, and Min Wu. *Dialogue management for an automated multilingual call center*. Tech. rep. Institute for Informatics, Logics and Security Studies, University at Albany, NY., 2003.

[19] Hunter Heidenreich. *What is a Generative Adversarial Network?* URL: https://towardsdatascience.com/what-is-a-generative-adversarial-network-76898dd7ea65.

[20] Matthew Henderson, Blaise Thomson, and Jason D Williams. "The second dialog state tracking challenge". In: *Proceedings of the 15th annual meeting of the special interest group on discourse and dialogue (SIGDIAL)*. 2014, pp. 263–272.

[21] Jonathan Hui. *GAN — Energy based GAN (EBGAN) Boundary Equilibrium GAN (BEGAN)*. URL: https://medium.com/@jonathan_hui/gan-energy-based-gan-ebgan-boundary-equilibrium-gan-began-4662cceb7824.

[22] Jonathan Hui. *GAN — LSGAN (How to be a good helper?)* URL: https://medium.com/@jonathan_hui/gan-lsgan-how-to-be-a-good-helper-62ff52dd3578.

[23] Jonathan Hui. *GAN — RSGAN RaGAN (A new generation of cost function.)* URL: https://medium.com/@jonathan_hui/gan-rsgan-ragan-a-new-generation-of-cost-function-84c5374d3c6e.

[24] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. "Image-to-image translation with conditional adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[25] Ridong Jiang, Yeow Kee Tan, Dilip Kumar Limbu, Tran Anh Dung, and Haizhou Li. "Component pluggable dialogue framework and its application to social robots". In: *Natural Interaction with Robots, Knowbots and Smartphones*. Springer, 2014, pp. 225–237.

[26] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[27] Kyungduk Kim, Cheongjae Lee, Sangkeun Jung, and Gary Geunbae Lee. "A frame-based probabilistic framework for spoken dialog management using dialog examples". In: *Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue*. 2008, pp. 120–127.

[28] Margarita Kotti, Vassilios Diakoloukas, Alexandros Papangelis, Michail Lagoudakis, and Yannis Stylianou. "A Case Study on the Importance of Belief State Representation for Dialogue Policy Management." In: *INTERSPEECH*. Vol. 986. 2018.

[29] Michail G Lagoudakis and Ronald Parr. "Least-squares policy iteration". In: *Journal of machine learning research* 4.Dec (2003), pp. 1107–1149.

[30] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. "Adversarial learning for neural dialogue generation". In: *arXiv preprint arXiv:1701.06547* (2017).

[31] Lihong Li, Jason D Williams, and Suhrid Balakrishnan. "Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection". In: *Tenth Annual Conference of the International Speech Communication Association*. 2009.

[32] J. Liang and R. Liu. "Stacked denoising autoencoder and dropout together to prevent overfitting in deep neural network". In: *2015 8th International Congress on Image and Signal Processing (CISP)*. 2015, pp. 697–701.

[33] Fotios Lygerakis. *Belief state space representation for statistical dialogue managers using deep autoencoders*. Diploma Thesis. School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece, 2019.

[34] Fotios Lygerakis, Vassilios Diakoloulas, Michail Lagoudakis, and Margarita Kotti. "Robust Belief State Space Representation for Statistical Dialogue Managers Using Deep Autoencoders". In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE. 2019, pp. 1055–1061.

[35] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. "Adversarial autoencoders". In: *arXiv preprint arXiv:1511.05644* (2015).

[36] Håkan Melin, Anna Sandell, and Magnus Ihse. "CTT-bank: A speech controlled telephone banking system-an initial evaluation". In: 1 (2001), pp. 1–27.

[37] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. "Unrolled generative adversarial networks". In: *arXiv preprint arXiv:1611.02163* (2016).

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[39] Naresh Nagabushan. *A wizard's guide to Adversarial Autoencoders: Part 2, Exploring latent space with Adversarial Autoencoders.* URL: https://towardsdatascience.com/a-wizards-guide-to-adversarial-autoencoders-part-2-exploring-latent-space-with-adversarial-2d53a6f8a4f9.

[40] Parul Pandey. *Understanding the Mathematics behind Gradient Descent.* URL: https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e.

[41] Benedetto Piccoli and Francesco Rossi. "Generalized Wasserstein distance and its application to transport equations with source". In: *Archive for Rational Mechanics and Analysis* 211.1 (2014), pp. 335–358.

[42] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[43] Sai Rajeswar, Sandeep Subramanian, Francis Dutil, Christopher Pal, and Aaron Courville. "Adversarial generation of natural language". In: *arXiv preprint arXiv:1705.10929* (2017).

[44] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. "Generative adversarial text to image synthesis". In: *arXiv preprint arXiv:1605.05396* (2016).

[45] Nicholas Roy, Joelle Pineau, and Sebastian Thrun. "Spoken dialogue management using probabilistic reasoning". In: *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 2000, pp. 93–100.

[46] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[47] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans". In: *Advances in neural information processing systems*. 2016, pp. 2234–2242.

[48] Pei-Hao Su, Pawel Budzianowski, Stefan Ultes, Milica Gasic, and Steve Young. "Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management". In: *arXiv preprint arXiv:1707.00130* (2017).

[49] Stefan Ultes, Lina M Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Inigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, and Milica Gasic. "Pydial: A multi-domain statistical dialogue system toolkit". In: *Proceedings of ACL 2017, System Demonstrations*. 2017, pp. 73–78.

[50] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *Journal of machine learning research* 11.Dec (2010), pp. 3371–3408.

[51] Zhuoran Wang, Tsung-Hsien Wen, Pei-Hao Su, and Yannis Stylianou. "Learning domain-independent dialogue policies via ontology parameterisation". In: *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. 2015, pp. 412–416.

[52] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[53] Lilian Weng. *From Autoencoder to Beta-VAE*. URL: https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html.

[54] Jason D Williams and Steve Young. "Scaling POMDPs for spoken dialog management". In: *IEEE Transactions on Audio, Speech, and Language Processing* 15.7 (2007), pp. 2116–2129.

[55] Jason D Williams and Steve Young. "Scaling up POMDPs for Dialog Management: The"Summary POMDP"Method". In: *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.* IEEE. 2005, pp. 177–182.

[56] Steve Young. "Talking to machines (statistically speaking)". In: *Seventh International Conference on Spoken Language Processing*. 2002.

[57] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. "Pomdp-based statistical spoken dialog systems: A review". In: *Proceedings of the IEEE* 101.5 (2013), pp. 1160–1179.

[58] Steve Young, Jost Schatzmann, Karl Weilhammer, and Hui Ye. "The hidden information state approach to dialog management". In: *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*. Vol. 4. IEEE. 2007, pp. IV–149.