Belief State Space Representation for Statistical Dialogue Managers Using Deep Autoencoders

Fotios Lygerakis

A thesis presented for the degree of Integrated Master of Engineering

Committee:

Professor Vassilios Digalakis
Associate Professor Michail G. Lagoudakis
Dr. Vassilios Diakoloukas

School of Electrical and Computer Engineering
Technical University of Crete
Greece
June 2019

Abstract

Statistical Dialogue Systems (SDS) have proved their humongous potential over the past few years. However, the lack of efficient and robust representations of the belief state (BS) space refrains them from revealing their full potential. Furthermore, there is a great need for automatic BS representations, which will replace the old hand-crafted, variable-length ones. To tackle those problems, we introduce a novel use of the Autoencoder (AE). Our goal is to obtain a low-dimensional and compact, yet robust representation of the BS space. We investigate the use of dense AE, Denoising AE (DAE), Sparse Denoising AE (SDAE) and Variational Denoising AE (VDAE). Denoising approaches are particularly useful when corrupted BS vectors exist in environments with high uncertainty. We also explore the capabilities of Domain Independent Parameterization (DIP), a domain-independent alternative to the BS representation, and we combine them with different types of the AE, in order to obtain a more compact and robust representation of it. The BS space representation obtained from the AE is then used by two state-of-the-art Reinforcement Learning (RL) algorithms to learn the dialogue policies from simulated users in the PyDial toolkit; the non-parametric GP-SARSA and the parametric LSPI. In this framework, the BS is normally represented in a relatively compact, but still redundant summary space which is obtained through a heuristic mapping of the original master space. We show that the proposed AE-based representations consistently outperform the summary BS representation. Especially, as the Semantic Error Rate (SER) increases, the DAE/VDAE-based representations obtain state-of-the-art and sample efficient performance.

Contents

Chapter 1

Introduction

In the past few years, there has been a rising interest in incorporating Spoken Dialogue Systems [31, 40, 12] in real-life applications. Apple Siri, Amazon Alexa, Microsoft Cortana and Google Assistant are few examples of novel commercial systems that use voice commands to control multiple devices at a place or can be asked for information. Many industries, like social robots [11], tourist information [2], banks [26], navigation [15], health care [28] and calling centers [9], have already incorporated Spoken Dialogue Systems to improve performance and offer better services. At its core, a Dialogue System utilizes a Dialogue Manager (DM) [41], which keeps track of the dialogue's state at its State Tracker and predicts an action based on that state at the Policy Manager (PM). So far, rule-based DMs [3] have shown great performance in many domains, but their building complexity, cost of scalability, sensitivity to semantic errors, need for continuous supervision and domain-restricted usage have turned the research interest towards the development of Statistical DMs (SDMs) [41]. Their growth of popularity is a consequence of the numerous training corpora that became available online in the past few years, the fact that they enable the introduction of a continuous state space to the models and their generalization capability.

There are PMs for the SDMs based on:

- *non-parametric* RL algorithms, i.e. the Gaussian Process SARSA (GP-SARSA) [7]
- *linear parametric* RL algorithms, i.e. the Least-Squares Policy Iteration (LSPI) [20][22]
- *non-linear parametric* RL algorithms, such as the Deep Q-learning Network (DQN) [27], the Advantage Actor Critic (A2C) [6] and the episodic Natural Actor Critic (eNAC) [33]

In [3] there is a comparison between those algorithms on different simulated environments and domains in which GP-SARSA seems to outperform all the other deep Neural Network (DNN) approaches. For the benchmarking procedure they use the summary BS (sumBS) representation [38, 39] to train the policy managers. The sumBS space is a heuristic mapping of the master (full) BS space. However, since

it uses the domain ontology the sumBS vectors produced are domain-dependent, relatively high-dimensional, sparse and redundant. They also require domain expertise and they cannot be easily generalized and adapted to new domains. Those characteristics of the sumBS vectors prevent RL algorithms from finding good policies in reasonable time. For example, LSPI [20] having a cubic complexity on the dimensions of the BS vector, is highly dependent on the size of sumBS vector.

Furthermore, in [3] the performance of the dialog manager degrades significantly in the presence of noise or when the domain complexity increases. This performance degradation could be attributed to the highly-dimensional and redundant sumBS vector. We believe that a more efficient BS representation could result in large performance gains particularly in difficult environments.

In this thesis we investigate robust and efficient BS space representations automatically obtained from AE variations [19]. Although AE variations have been successfully used for non-linear feature extraction at several application domains, to the best of our knowledge, this is the first time that they are used as a tool to project the fullBS space into a more compact and lower-dimensional space in the framework of a SDS. We also examine a noise robust variation of the AE; the Denoising AE (DAE) [36]. DAE can overcome the sensitivity of other featurizations in environments that introduce high SER. Complementary, we investigate the use of Sparse DAE (SDAE) based on [25], hoping that it will provide distinct representations and enable PM to learn more robust policies. We also introduce another promising AE-based representation technique, namely Variational AE (VAE) [16] [8] and Variational DAE (VDAE). VAE/VDAE manage to capture the true distribution that creates the latent space representation of the sumBS space, performing variational inference on the learned distribution to obtain the sumBS representations.

In addition, we examined the efficiency of the AE in other feature vectors. Specifically, we considered the Domain Independent Parameterization (DIP) [37], developed by "Toshiba Research Europe LTD" which defines a smaller domain-independent state representation. Despite its smaller size, the DIP representation is redundant, and exhibits high dependency among its features. We therefore utilize DAE and VDAE to further reduce its size and obtain a better and robust representation.

Finally, we use the above representations to learn optimal dialogue policies with two different state-of-the-art RL algorithms: a parametric and a parameter-free one, namely LSPI [20] and GP-SARSA [7] respectively. We describe a training scheme which concurrently adjusts the parameters of the AE, DAE, SDAE or VDAE networks with new episodes that are also used for policy optimization (GP-SARSA, LSPI). We confirm the high performance of the proposed schemes over a series of experiments for several domains in multiple noise environments, using the PyDial dialog simulation toolkit.

This thesis is organized in the following chapters: In Chapter 2 we present the general modular approach of Dialogue Systems, the mathematical background and available tools. We then explain the constraints and we discuss how they are approached in the literature. In Chapter 3 we describe the AE and its different varia-

tions we use, as well as DIP features and our training scheme and in Chapter 4 we show and discuss the results of our experiments. Chapter 5 concludes our work and describes possible extensions in the future.

Chapter 2

Background

2.1 Dialogue Management

The basic architecture of a Spoken Dialogue System [41] consists of the Spoken Language Understanding (SLU) module, the DM and a natural language generation (NLG) module (Figure 2.1). We assume that a dialogue consists of turns between

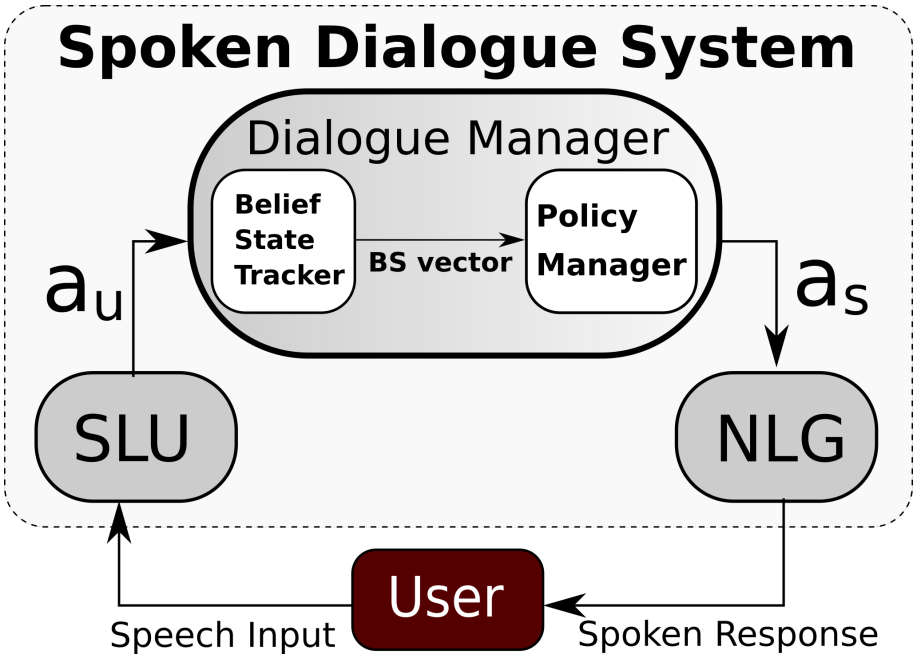


Figure 2.1: Modular Architecture of a Dialogue System based on [41].

the user and the system. At each turn, the user gives a speech input to the SDS. In the SLU module, an automatic speech recognizer converts the user's speech input to text and the natural language understating component translates text into an abstract semantics representation in the form of user's intention or user act a_u. The belief state tracker in the DM makes use of a_u to estimate the current belief state of the dialogue, providing an updated BS vector to the PM to predict the next system action a_s. Finally, in the NLG module the system action a_s is translated into natural language text through a template-filling procedure and a text-to-speech

module returns a spoken system response to the user in spoken language, based on the result of the template-filling.

2.2 POMDPs

Managing a dialogue can be seen as a discrete-time sequential decision process and, thus, is generally approached as a Markov Decision Process (MDP) problem [1]. An MDP is based on the Markov's property, which states that *"A future state is independent of all the past states given the present one"*. Specifically, the current state where we are at is a sufficient statistic to predict a future one. We describe an MDP as a 4-tuple:

$$(S, A, T_a, R_a) \tag{2.1}$$

where S is a finite set of the 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 as 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.

In reality though, due to the fact that there is no strict description of the environment's models, i.e. the models describing the behavior or goals of the user or the model that adds noise in the communication channel, the underlying state is partially observable. This partial observability is caused by the errors introduced to the user's input through the channel, mistakes at the semantics extraction and the uncertainty of the user's true goal. Thus, we can frame this problem as a discrete-time Partially Observable Markov Decision Process (POMDP) [41] [7].

In a POMDP, at each step t of a dialogue, the DM maintains a distribution over the set of all possible states S, performs an action $a \in A$ using the derived policy π, moves in the new (unobserved) state $s' \in S$, receives a corresponding reward r equals to the function $R : S \times A \rightarrow \mathbb{R}$ while taking the observation $\omega \in \Omega$ and proceeds repeatedly. To have a complete description of a POMDP we need the set of conditional transition probabilities between states T, the set of conditional observation probabilities O and the discount factor $\gamma \in [0, 1]$. Therefore, a POMDP is the $7 - tuple$ [13]:

$$(S, A, T, R, \Omega, O, \gamma) \tag{2.2}$$

The only information the agent can acquire from a dialogue environment is a potentially delayed reward signal r and the observation of the state o. We can use o at the belief tracker of the DM to formulate the BS vector $\boldsymbol{b} \in B : \mathbb{R}^{|S|}$, which is a vector that keeps a distribution over all the possible states s and use it instead of s to cope with the uncertainty of observations.

The policy manager's goal is to find a policy π of actions a at each time step t, such that $a_t = \pi(\boldsymbol{b_t})$, that maximizes the discounted expected future reward starting from the initial belief $\boldsymbol{b_0}$:

$$V^{\pi}(\boldsymbol{b_0}) = \sum_{t=0}^{\infty} \gamma^t E\left[R(\boldsymbol{b_t}, \boldsymbol{a_t}) | \boldsymbol{b_0}, \pi\right] \tag{2.3}$$

Then optimal policy $\pi*$ is obtained as:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} V^{\pi}(b_0) \tag{2.4}$$

For each belief state $\boldsymbol{b}$ we can find π^* by obtaining the value function V^* for each $\boldsymbol{b}$. We can obtain V^* as the solution to the Bellman optimality equation:

$$V^*(\boldsymbol{b}) = \max_{a \in A} \left[r(\boldsymbol{b}, a) + \gamma \sum_{o \in O} Pr(o|\boldsymbol{b}, a) V^*(\mathrm{T}(\boldsymbol{b}, a, o)) \right] \tag{2.5}$$

where T is the belief state transition function.

At this point we could associate the full BS vector and the corresponding action derived from the DM policy module with environment reward and respectively use reinforcement learning (RL) to learn optimal decision policies.

Unfortunately, in real-world problems we cannot acquire the transition function T, so we cannot use dynamic programming, value or policy iteration to find the optimal policy. In addition, in these applications the full BS and action spaces are very large and probably continuous. This introduces more barriers in finding optimal policies as the solutions to the above equations become intractable. Later in this chapter we are explaining how these barriers can be broken by the state-of-the-art and our proposed technique in Chapter 3.

2.3 Reinforcement Learning

To avoid the need to acquire or compute T from 2.2 we have to resort to model-free RL algorithms. Q-learning is such an algorithm at which, instead of the value function $V : B \to \mathbb{R}$, we calculate $Q : B \times A \to \mathbb{R}$. The algorithm updates Q at each time step t as:

$$Q(\boldsymbol{b_t}, a_t) \leftarrow Q(\boldsymbol{b_t}, a_t) + \alpha \left(r_t + \gamma \max_a Q(\boldsymbol{b_{t+1}}, a_{t+1}) - Q(\boldsymbol{b_t}, a_t) \right) \tag{2.6}$$

where α is the learning rate, $Q(\boldsymbol{b_t}, a_t)$ the old value and $Q(\boldsymbol{b_{t+1}}, a)$ the estimation of the optimal future reward. In this way, the Q-function is only dependent on the $4 - tuple$:

$$(\boldsymbol{b}, a, r, \boldsymbol{b'}) \tag{2.7}$$

where $\boldsymbol{b'}$ and $\boldsymbol{b}$ are the current and previous belief states respectively, $(\boldsymbol{b}, \boldsymbol{b'}) \in B$, $a \in A$ and $r \in \mathbb{R}$.

Q-learning is an off-policy algorithm, meaning that it does not follow a specific policy π to find the next action. In fact, this algorithm chooses the next action in a greedy way, selecting the one that provides the highest reward. For this reason, Q-learning optimization is performed on-line while interacting directly with the user.

Another RL algorithm that is model-free and can be trained on-line is State-Action-Reward-State-Action (SARSA). SARSA algorithm is an on-policy algorithm, meaning that it updates the policy π while interacting with the environment, i.e.

updates π while using it, which is its only difference comparing it with Q-learning. Likewise, we use equation 2.6 to update the value function Q.

Nevertheless, Q-learning and SARSA are not suitable to be used in big-scale real-world applications, since they make use of matrices to store the value function data and, thus, cannot be used for large BS and action spaces, not even mentioning the cases where data are in $\mathbb{R}$. To overcome this boundary we have to approach the value function Q with function approximation techniques. The most prominent function approximators that have been tested so far for DMs are Gaussian Processes (GP) [7], linear parametric combinations of basis functions for Q (LSPI) [20] and Deep Neural Networks(DNNs)[27].

2.3.1 GP-SARSA

A Gaussian Process (GP) is a collection of random variables that are described by a multivariate normal distribution or, equivalently, a distribution over possible functions. A GP's distribution is the joint distribution of these random variables (functions), which can be infinite and additionally handle continuous values. Hence, a GP is a non-parametric function approximator, meaning that it has an infinite number of parameters. For GP approximation we start from the prior knowledge we have and we use Bayes' rule to update the distribution of function by observing training data [7].

The motivation on using GP to approximate the Q-function is quite obvious. The GP-based approximator enables Q to handle arbitrary high dimensional states and handle values in $\mathbb{R}$ altogether. If we substitute the update rule of 2.6 at the right hand of the Q correction with the update of the GP-based Q-function approximation we are obtaining the GP-SARSA algorithm.

2.3.2 Least Squares Policy Iteration

Least Squares Policy iteration (LSPI) [20] belongs to the family of approximate policy iteration RL algorithms, which learns decision policies from samples that are produced from the interaction with the environment, e.g. dialogue samples when talking to a user in the case of SDSs. The iteration starts with a parametric approximation of the Q-function with free weights w:

$$\tilde{Q}^{\pi}(\boldsymbol{b}, a; w) = \sum_{j=1}^{k} \phi_j(\boldsymbol{b}, a) w_j \tag{2.8}$$

The basis functions ϕ_j are fixed in size, which depends on the specific application and can be chosen appropriately and, generally, they are non-linear functions of $\boldsymbol{b}$ and a. Then LSPI proceeds to *policy improvement*, choosing greedily at each iteration m the action that maximizes Q^{π_m}:

$$\pi_{m+1}(\boldsymbol{b}) = \underset{a \in A}{\operatorname{argmax}}\, Q^{\pi_m}(\boldsymbol{b}, a) \tag{2.9}$$

Algorithm 1 LSTDQ algorithm with a model - Learns the approximate Q of a fixed policy [20]

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

    $\tilde{\boldsymbol{A}} \leftarrow \mathbf{0}$ ▷ Weights: $(k \times k) matrix$
    $\tilde{\boldsymbol{b}} \leftarrow \mathbf{0}$ ▷ 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 2 LSPI algorithm - Iteration between Policy Improvement and Policy Iteration [20]

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 $w_0 (usuallyequalsto 0)$

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

    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

and *policy evaluation* (Algorithm 1 [20]), where the Least Squares Temporal Difference Q-function (LSTDQ) algorithm is being used to learn the Q-function.

In Algorithm 1 [20]) we are using the state s instead of the belief state vector $\boldsymbol{b}$, so there is no confusion between the notation between the bias and the belief state vector. Without losing generalization capability we can use samples of the form $(\boldsymbol{b}, a, r, \boldsymbol{b'})$ at the LSTDQ algorithm. Finally, we use the LSPI algorithm (Algorithm 2 [20])) to find an optimal policy based on the iteration described above.

Algorithm 3 LSTDQ-OPT algorithm - Direct update on $\tilde{\boldsymbol{A}^{-1}}$ [20]

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

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

Is is important to note, that the LSTDQ algorithm as described in Algorithm 1 demands the computation of $\tilde{\boldsymbol{A}}^{-1}$ which has a cubical dependence ($O(k^3)$) on the number of basis functions k in terms of complexity. This would lead to bad system performance, especially if we want to update equation (2.8) frequently. A variation of LSTDQ would aim to update directly $\tilde{\boldsymbol{A}}^{-1}$ matrix, which is denoted as $\boldsymbol{B}$ in Algorithm 3 [20]. With this alteration we can easily and very cost-effectively update out linear system with only squared dependence on k ($O(k^2)$). Finally, like in the original LSTDQ algorithm, we are using the state s instead of the belief state $\boldsymbol{b}$ to eliminate confusability with the bias vector.

In this work we are using the Block basis functions. The block-basis feature is one of the simplest state-action encodings. Considering a dialog of n discrete dialog actions $A = (a_1, a_2, \ldots, a_n)$, the block-basis feature is constructed as a concatenation of n block vectors, each having the size of the BS vector. In this framework, the state-action representation is obtained by the activation of a single feature-vector block that corresponds to the selected action. All other blocks in the feature-vector are zero-valued. Specifically, let the BS vector be represented by a d-dimensional vector $\boldsymbol{s} = (s_2, s_2, \ldots, s_d)$. Then the block-basis feature for the state $\boldsymbol{s}$ and a corresponding

active action a is created as follows:

$$\phi(\boldsymbol{s}, a) = [c_1\boldsymbol{s}, c_2\boldsymbol{s}, \ldots, c_n\boldsymbol{s}]^T \tag{2.10}$$

where $c_i, i = 1, \ldots, n$ is a binary constant defined as:

$$c_i = \begin{cases} 1, & \text{if } a_i = a \\ 0, & \text{otherwise} \end{cases} \tag{2.11}$$

Although this can be an elegant representation for the dialog domain since it nicely discriminates the action contribution, it also exhibits several disadvantages. For instance, even a small to medium action space can result to sparse and high dimensional block-basis features for a typical belief state-vector. This results to large computational burdens during the policy learning process and makes the use of large action spaces impractical.

2.3.3 Non-Linear Approximators

A very popular Q-function approximation category, that has attracted attention and important research interest, is the DNN-based approximation techniques, i.e. DQN [27], A2C [6] and eNAC [33]. The basic concept here is that we want to learn a function that takes as input the state of the dialogue at each time step and predicts the value of the Q-function.

The prominence of this technique arises from the ability of these approximators to capture really complex dependency structures of the dialogue state at each time step. This is promoted by the non-linearity that is being imposed to the sub-functions consisting the approximator of the Q-function, also known as Neural Network, as well as the architecture of its topology.

2.4 Belief State Space Representation

2.4.1 Full Belief State Space

As we discussed in Section 2.2 of this Chapter, we keep a distribution over all possible states due to the uncertainty in our environment. An example of such dialogue states at a domain on Laptops purchase is:

- *inform*(type="Laptop", price="medium",HDD="750G", cpu="2.2Ghz", warrranty="2years",...,type="business")(0.15)

- *inform*(type="Notebook", price="cheap",HDD="500G",cpu="2.2Ghz", warrranty="3years",...,type="basic")(0.07)

- ...

In parenthesis after the state description is the probability of being in that state. It can be easily inferred from the above example that the number possible states, in which a dialogue can be at, has exponential dependence on the number of the available values of each feature in the ontology. For example, if we consider 7 features in the ontology of a domain with an average cardinality of |10| each, we end up with $10^7 = 10$ million states.

2.4.2 Summary Space

An early tool that enabled low dimensional representations of the full BS space, that could empower PM to learn better policies more efficiently, was the creation of a summary space [38] [39] from the original fullBS space. Williams and Young in [38] call the fullBS space the master space. Thus, summary space is a subset of the master one and PM operates only on this subset, which leads to tractable computations for finding optimal policies, since during an average dialogue the number of states it is usually visited is a small subset of all the available ones.

In [42], the summary space is formed by factorizing the master space into the user's goal state space S_u^w and dialogue's history state space S_d^w for every slot w from the set of slots W that each of these two spaces is decomposed at. The result is two new summary space sets $\hat{S}_u^w = \{best, rest\}$ and $\hat{S}_d^w = S_u^w$. The mapping from the master to the summary space is being done by setting each belief component equal to the probability mass of the most likely user goal in slot w ($\hat{b}(\hat{s}) = best$) and the dialogue history equal to the history of the master space ($\hat{b}((\hat{s}_d^w)) = b(\hat{s}_d^w)$)

Thus, in PyDial the sumBS vector is formed as a dictionary that contains domain-dependent and domain-independent beliefs, domain-independent features and domain-dependent user acts as three distinct sub-dictionaries:

- *beliefs*:
  - domain-independent: *"discourseACT"* and *"method"*.
  - domain-dependent. Example: *'pricerange'* and *"batteryrating"* (Laptops11 domain)
- *features*:
  - *"inform_info"*
  - *"informedVenueSinceNone"*
  - *"lastActionInformNone"*
  - *"offerHappened"*
- *userActs*
  - *inform(weightrange="lightweight", isforbusinesscomputing="1")*
  - *affirm(driverrange="small")*

Each slot of the above sub-dictionaries can take any type of values depending on the ontology; binary or decimal values, discrete or continuous numbers and strings. It is important to mention that the domain-dependent beliefs are based on the ontology of each domain.

To update the sumBS vector we use the Focus Tracker from [10], which is implemented in PyDial. The tracker associates a probability in each slot, based on the hypothesis's of the SLU components of the SDS, and outputs the updated sumBS vector. Finally, this dictionary is flattened into a vector that contains only the values of the aforementioned dictionaries, i.e. a variable size vector of probabilities (Tables 2.1 and 2.2).

1	**2**	**3**	**...**	**268**
discourseAct: "ack"	pricerange: "expensive"	offerHappened: "true"	...	inform (cuisine="greek", kidsfriendly="1")
0.046	**0.953**	**0.000**	**...**	**1.000**

Table 2.1: Cambridge Restaurants domain sumBS vector example.

1	**2**	**3**	**...**	**257**
discourseAct: "hello"	pricerange: "cheap"	offerHappened: "true"	...	inform (weightrange="lightweight", isforbusinesscomputing="1")
1.000	**0.000**	**0.095**	**...**	**1.000**

Table 2.2: Laptops11 domain sumBS vector example.

The obtained flattened sumBS vector is therefore domain-dependent, sparse, still relatively high-dimensional and information-redundant and it refrains most PMs from learning good policies or from converging fast. Consequently, there is a great need to represent the sumBS space into a more efficient and compact one.

2.4.3 Domain-Independent Parameterisation

Another approach towards efficient and domain-independent BS space is the Domain Independent Parameterization (DIP) [37]. DIP produces a BS space based on the domain-independent information from the SLU module and statistics on the domain-dependent ones. The result is a low-dimensional, fixed size feature vector, that can be seamlessly used in multiple domains. DIP exploits the underlying similarities and the encapsulated information of task in different domains and extracts features based on grouping relevant slots in the domain ontologies, on the history of the dialogue and on the last action. These features, however, need to be hand engineered by the system designer. Specifically, DIP state vector might include redundant, correlated or non-informative features, they do not make use of domain-specific information

and are sensitive to noise. Finally, this vector may vary in size depending on the number of features we want to use.

2.4.4 Binary Feature Representation

Another alternative BS representation is the BinLin/BinAux BS introduced in [18], which also uses limited domain information to build extremely compact state representations. BinLin/BinAux are based on the assumption that the policy manager's decision, on what the next action will be, is dependent on the presence or not of a slot in the ontology. Thus, binary features actually represent the activation of a slot value for the given turn and they exclude its specific value. The featurization of the BS space is achieved either by obtaining a linear combination(BinLin) for all the domain slots, or by augmenting them with auxiliary binary information about slot requests(BinAux). However, they need to be explicitly defined for each domain and are sensitive to the input's uncertainty and their performance degrades in the presence of noisy semantic input.

2.4.5 Non-Linear Automatic Feature Representations

In recent work in [5, 4], a feed forward network (FNN) and a recurrent neural network (RNN) are used to learn feature extractors in the form of feature functions for each slot to obtain abstractions. of the BS space. However, this approach does not seem to clearly outperform the techniques mentioned above and the resulting system is prone to slow and non-optimal converge.

In [22] [23] feature selection is used to obtain lower-dimensional state representation. However, most of the algorithms are tied to specific RL algorithms such as LSPI, and cannot be generalized to other policy models. Furthermore, they do not exhibit robustness in the presence of noise.

Chapter 3

Non-Linear Belief State Representation

3.1 Deep Neural Networks

The building unit of a neural network (NN) is the neuron. A neuron is a mathematical function that takes an input and produces an output. Inspired by the biological neurons, the artificial one involves the multiplication of its input with a weight and the application of an activation function to the weighted input to produce the output y:

$$y = a(w \times x) + b \tag{3.1}$$

where $w \in \mathbb{R}_1$ denotes the neuron's weight, $\alpha()$ defines the activation function, $x \in \mathbb{R}^1$ is the input of the neuron and $b \in \mathbb{R}_1$ is the bias. In this thesis we are excluding bias, without losing the ability of generalization.

Choosing the activation function appropriately, we can transform the neuron input non-linearly. This non-linearity can be introduced with a sigmoid-shaped function, like sigmoid, tanh and softmax, or the Rectified Linear Unit (ReLU). Non-linearity enables neural networks to approximate complex functions that traditional linear models are incapable of. Multiple neurons can be stacked together into a layer and handle a vector of real values $\boldsymbol{x} \in \mathbb{R}^{d_x \times 1}$, where d_x are the dimensions of $\boldsymbol{x}$. Then 3.1 becomes:

$$y(\boldsymbol{x}) = a\left(W \times \boldsymbol{x}\right) \tag{3.2}$$

or

$$y_k(\boldsymbol{x}) = a\left(\sum_{i=0}^{d_z} w_{ki} \times x_i\right) \tag{3.3}$$

where $y(\boldsymbol{x}) \in \mathbb{R}^{d_y \times 1}$, y_k is the k-th neuron of the layer $\boldsymbol{y}$ and $W \in \mathbb{R}^{d_y \times d_0}$.

A NN may consist of multiple layers of neurons connected together, formulating a directed acyclic graph. We call $\boldsymbol{x}$ the input layer and $\boldsymbol{z}$ the output layer of the NN. In addition, we can interpolate between the input and output layers one or more hidden layers, creating a Deep NN (DNN). For the simplest DNN, we can consider only one hidden layer $h_1(\boldsymbol{x})$ and the weight matrix $W_{h_1 x} \in \mathbb{R}^{d_{h_1} \times d_x}$ defines

the matrix of weights connecting the input and the first hidden layer:

$$h_1(\boldsymbol{x}) = \alpha\left(W_{h_1 x} \times \boldsymbol{x}\right) \tag{3.4}$$

Then, with $W_{yh_1} \in \mathbb{R}^{d_y \times d_{h_1}}$, the output layer $y(\boldsymbol{x})$ is calculated as:

$$y(\boldsymbol{x}) = \alpha\left(W_{zh_1} \times h_1(\boldsymbol{x})\right) \tag{3.5}$$

Several hidden layers $j \in 1, \cdots, N$, can then be added with varied dimensions depending on the architecture. The corresponding hidden representation of a hidden layer j is then defined as a mapping from the previous layer $j-1$ as follows:

$$h_j(\boldsymbol{x}) = \alpha\left(W_{h_j h_{j-1}} \times h_{j-1}(\boldsymbol{x})\right) \tag{3.6}$$

The most common way to train a NN is through backpropagation [21]. A forward pass from the input to the output layer is computed to obtain the prediction $\boldsymbol{y} = y(\boldsymbol{x})$ at the output layer. Then we can compute the loss J based on the error between the predicted value and the label $\boldsymbol{l}$ that is the target of the NN. For example a popular loss function is the mean square error:

$$J(\boldsymbol{y}, \boldsymbol{l}) = \mathbb{E}\left[||\boldsymbol{y} - \boldsymbol{l}||^2\right] \tag{3.7}$$

Consequently, we backpropagate J to train the weights of the NN, using a fitting optimizer [30], i.e. Gradient Descent [34] and ADAM [17]. In the most applications NNs are used either for classification [45] or regression [32] and are trained in a supervised learning framework, i.e. we have in our possession a set of labelled data and we compute the loss based on the difference between the prediction of the data from the NN and the true labels of the input. However, in this thesis we are considering a specific NN topology, the AE, in the framework of unsupervised learning in order to extract useful features from the input $\boldsymbol{x}$.

3.2 Vanilla Autoencoder

AEs are a family of NN topologies used for unsupervised learning which have been successfully employed for non-linear feature extraction at several application domains [29, 19, 24]. Due to its architecture, an AE is forced to learn lower dimensional and more robust representations of the input vector $\boldsymbol{x}$. In this thesis, input $\boldsymbol{x} \in \mathbb{R}^{d_x \times 1}$ denotes the sumBS vector (Section 2.4) which has a variable length d_x depending on the chosen domain. The AE is then used to project the sumBS from the $\mathbb{R}^{d_x \times 1}$ space into a fixed-size, lower-dimensional one.

Typically, an AE can be seen as the concatenation of two networks (Figure 3.1). The first one is the encoder, which projects the input layer to a lower-dimensional latent space. The second one is the decoder, which takes the encoded representation and projects it back to the original input. Their architecture exhibits a perfect symmetry with regards to the central hidden layer which is the lowest-dimensional one and serves as the coding layer.

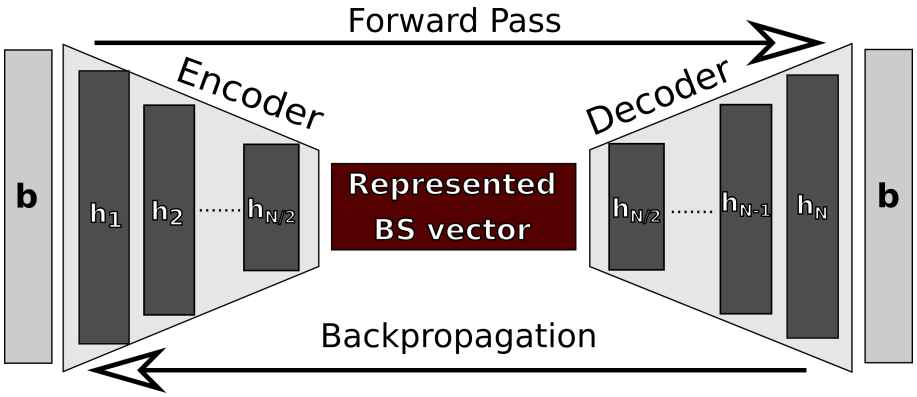


Figure 3.1: Encoder - Decoder Architecture of an Autoencoder

In this thesis, the AEs are built as deep, feed-forward and dense NNs consisting of multiple layers, as we described them in Section 3.1. The hidden representation of the projection layer $h_{\lceil \frac{N}{2} \rceil}$ is obtained through a typical mapping of the input, as described in Equation 3.6 using the encoders weight matrices W^{enc}. The latent representation $\boldsymbol{z}$ (Equation 3.8) from the bottleneck layer $h_{\lceil \frac{N}{2} \rceil}$ is then mapped back to the input $\boldsymbol{x}$ through a reverse mapping function using the corresponding weight matrices of the decoder W^{dec}.

$$\boldsymbol{z} = h_{\lceil \frac{N}{2} \rceil}(\boldsymbol{x}) \tag{3.8}$$

In this work, the decoder's weight matrices were tied to the corresponding encoder's weight matrices $W^{dec} = [W^{enc}]^T$.

3.3 Denoising Autoencoder

Originally the DAE was used to discard the noise from corrupted images [36]. The latent space in this case was higher-dimensional than the input. However, in this work, we keep the network's symmetry and the gradually smaller dimensions of the latent space as described in the previous Section 3.2, since our goal is to obtain also a compact representation.

In this thesis, DAE is essentially a dense AE which is trained to represent the corrupted or noisy input vectors, providing robustness. In a typical SDS, noise can be introduced due to a multitude of factors, including the errors of the recognizer, the semantic errors (e.g. acoustic confusability, ambiguity of natural language, incomplete utterances, etc.), as well as the uncertainty of user's goal. In specific, the input of a DAE is a corrupted version $\widetilde{\boldsymbol{x}}$ of the clean fullBS vector $\boldsymbol{x}$:

$$\widetilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{n} \tag{3.9}$$

where $\boldsymbol{x}$ is the target in the output layer, and $\boldsymbol{n}$ denotes the noise vector, produced by an unknown distribution, which has the same size as the clean fullBS $\boldsymbol{x}$.Noise $\boldsymbol{n}$ is added to $\boldsymbol{x}$ artificially using the SER percentage as probability P_{SER}. Specifically, the corrupted vector $\widetilde{\boldsymbol{x}}$ is obtained by keeping, with probability $1 - P_{SER}$, the true semantic information for a slot, and making random selections from all the available

values in the ontology, with probability P_{SER}. It is worth noting that this artificial corruption process can be applied to data from both real and simulated dialogues.

Using an adequately large set of $[\boldsymbol{x}, \widetilde{\boldsymbol{x}}]$ sample pairs, a DAE can be trained to approximate the noise distribution and appropriately filter out the noise. The parameters of the DAE are trained so as to minimize the average reconstruction error J defined as the mean squared error:

$$J(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}\left[||\boldsymbol{x} - \boldsymbol{y}||^2\right] \tag{3.10}$$

where $\boldsymbol{y}$ is the prediction obtained at the output of the network's decoder when we input the noisy $\widetilde{\boldsymbol{x}}$, and $\boldsymbol{x}$ is the original non-corrupted vector.

3.4 Sparse Autoencoder

Another AE variation is the Sparse AE (SAE). SAE have proved to be very successful at several application domains such as in text extraction from images [25], locomotion [44] and facial expression recognition [43]. A SAE can be designed by introducing a sparsity penalty, so that the NN learns sparse representations of the input. This can potentially help policy managers to learn better policies, since the representation learned by the sparse autoencoder (SAE) contains a small number of neurons that are highly activated at a specific state and the rest that are close to zero. In detail, what we try to achieve at a SAE is an average activation across of the latent space that is close to a very small value ρ, called the sparsity parameter. This results in enforcing the SAE to learn to activate only a few neurons at a time, imposing the desired sparsity. To accomplish sparsity we introduce a sparsity penalty to the loss function, weighted by a parameter β. We calculate the average activation $\hat{\rho}$ of the latent space over all the samples m for each neuron k of the latent hidden layer $h_{\lceil \frac{N}{2} \rceil}(\boldsymbol{x})$ as:

$$\hat{\rho}_k = \frac{1}{m} \sum_{i=1}^{m} z_{ik} \tag{3.11}$$

where z_{ik} is the activation of the k-th neuron of the bottleneck layer $h_{\lceil \frac{N}{2} \rceil}$ for the i-th sample of the dataset.

Then, we enforce the mean value of the elements of $\hat{\rho}$ to take a value close to the sparsity parameter ρ. This procedure will highly discriminate the few relevant neurons weakening the inclusion of the irrelevant ones from the training procedure of the policy manager. To achieve this we treat ρ and $\hat{\rho}$ as the means of respectively two Bernoulli random variables and we calculate the Kullback-Leibler(KL) divergence between them. KL-divergence is a trivial method to measure the similarity of two different distributions.

$$\sum_{k=1}^{d_z} \rho \log \frac{\rho}{\hat{\rho}_k} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_k} \tag{3.12}$$

or

$$\sum_{k=1}^{d_z} KL(\rho || \hat{\rho}_k) \tag{3.13}$$

Eventually, we incorporate this penalty term in J using a weighting parameter β as:

$$J_{sparse} = J + \beta \sum_{k=1}^{d_{h_z}} KL(\rho||\hat{\rho}_k) \tag{3.14}$$

In [25, 44, 43], the the encoder has a higher-dimesnional output than the input in order to learn the desired sparse representations. In our work we are following the bottleneck architecture we applied for the vanilla AE and DAE, because we want to obtain a compact and low-dimensional representation. Finally, we are training our networks using the denoising technique described in Section 3.3 obtaining a Sparse DAE (SDAE).

3.5 Variational Autoencoder

All the previous variations of the AE suffer from the same weakness of their topologies. This is the poor generalization capability to unseen data, particularly considering the sparse and continuous nature of the sumBS vector in the input, caused by discontinuous nature of the learned latent space. A discrete latent space is characterized by bad interpolation of the samples and, consequently, the PM cannot learn good policies from representations of unseen states.

To overcome this problem we are investigating the use of the Variational AE (VAE) [16, 8]. The term *variational* is originated by the variational inference we are performing at the posterior distribution that describes the latent space. From a probability model perspective, a VAE contains the probability models of the $p(x)$ of observed data, also called *evidence*, x and $p(z)$ of the true latent variables z, as shown in Figure 3.2. The joint distribution of these two models is $p(x, y)$ using the

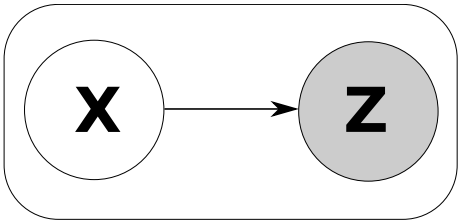


Figure 3.2: Graphical Representation of the dependence between data x and latent variable z.

Bayes' rule is:

$$p(x, z) = p(x|z)p(z) \tag{3.15}$$

and the posterior of the z given the data x:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{3.16}$$

However, in order to compute the prior probability of evidence $p(x)$ we have to marginalize out z from $\int p(x|z)p(z)dz$. As this requires the evaluation of all configurations of the latent variable z and, thus, exponential time, we have to approximate

the posterior probability $p(x|z)$. In variational inference we approximate $p(z|x)$ with a family of distributions $q(z|x;\boldsymbol{\lambda})$ and we learn the parameters $\boldsymbol{\lambda}$ with the help of an AE.

A VAE creates continuous latent spaces, since the latent vectors $\boldsymbol{z}$ are generated randomly from a parametric inference model defined as a multivariate Gaussian distribution $q(\boldsymbol{z}|\boldsymbol{x}:\boldsymbol{\lambda}) = \mathcal{N}(\boldsymbol{z}:\boldsymbol{\mu},\boldsymbol{\sigma})$, where $\boldsymbol{\lambda} = \{\boldsymbol{\mu},\boldsymbol{\sigma}\}$, $\boldsymbol{\mu}$ is the mean vector of the Gaussian and $\boldsymbol{\sigma}$ is the corresponding vector of standard deviations. The $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ vectors are approximated through a corresponding pair of weight matrices $W_{\boldsymbol{\mu}}$ and $W_{\boldsymbol{\sigma}}$ in the bottleneck layer $h_{\lceil \frac{N}{2} \rceil}$, which are optimized as parameters of the neural network.

In fact, instead of the σ values we maintain the log values of σ, $\log(\sigma)$. The reason is that predicting $\log(\sigma)$ instead of σ introduces stability during the training procedure and makes computations easier. This is beneficial to the training process, as $\log(\sigma)$ can ensure the non-negativity of σ and the avoidance of computations with very small numbers, that σ usually takes, during optimization.

The objective of the AE's optimization now has slightly changed. We keep the objective of learning representations that produce outputs identical to the input, while though also clustering states as close as possible to the neighbouring clusters, yet distinctly enough. This results to smoother interpolation of data, leading to more robust representations. Thus, optimization of the VAE is made on the basis of the following loss function:

$$J_{VAE}(\boldsymbol{x},\boldsymbol{y}) = J(\boldsymbol{x},\boldsymbol{y}) + D_{KL}\left(q(\boldsymbol{z}|\boldsymbol{x}:\boldsymbol{\lambda})||p(\boldsymbol{z})\right) \tag{3.17}$$

where D_{KL} denotes the KL-Divergence among the true latent variable distribution $p(\boldsymbol{z})$ which is typically chosen to follow the standard Gaussian $\mathcal{N}(0,1)$ and the approximation $q(\boldsymbol{z}|\boldsymbol{x}:\boldsymbol{\lambda})$ learned by the encoder.

When we want to use the encoder to produce a representation of the sumBS vector taken from the learned distribution q, we sample from the multivariate Gaussian q, using a equally-dimensional sample vector $\boldsymbol{\epsilon}$ from a multivariate Normal distribution, as:

$$\boldsymbol{r} = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\sigma}_{(}\boldsymbol{x}) * \boldsymbol{\epsilon} \tag{3.18}$$

In the proposed Variational DAE (VDAE) scheme, the inference model $q(\boldsymbol{z}|\tilde{\boldsymbol{x}}:\boldsymbol{\lambda})$ learns to generate latent vectors from corrupted sumBS state vectors $\tilde{\boldsymbol{x}}$ that have been injected with noise using the method described in Section 3.3. In this way, the VDAE will be trained to be tolerant in noise.

3.6 Concurrent Training Scheme

In this work, both the policy manager and the parameters of the AE variation in use are optimized in parallel using batches of dialogue episodes. The AE variation take as input sumBS vectors $\boldsymbol{x}$, while the policy manager takes as input the AE's

bottleneck layer $h_{\lceil \frac{N}{2} \rceil}(\boldsymbol{x})$. Consequently, both the sumBS representations and the policy are dynamically adapted to the environment's variations.

To speed up the convergence of this parallel process we perform pre-training on the AE based on randomly generated sumBS vectors. During our experiments we noticed that the convergence of the above process depends on the initialization of the AE's network weights. Good initial estimates of the network weights result to faster optimization and better policies of the policy manager. The pre-training process is terminated when the loss function drops below a threshold. Further optimization is then performed on-line to dynamically adapt the models to the environment's uncertainty. This process is summarized in Algorithm 4.

Algorithm 4 Concurrent Training Scheme

```
1: AE.preTrain()
2: for iteration ∈ (1, NumOfDialogueBatches) do
3:     for Dialogue in DialogueBatch do
4:         DialSimulation.Start()
5:         while !(DialSimulation.Ended()) = True do
6:             Policy.saveToBatch(Episode)
7:             AE.saveToBatch(Episode)
8:             if AE.BatchisFull() = True then
9:                 AE.train()
10:    Policy.train()
11:    Policy.evaluate()
```

Chapter 4

Experiments

4.1 Experimental Set Up - Framework

For our experiments, we used the GP-SARSA policy algorithm in the PyDial [35] on simulated standard users. We also incorporated the optimized version of LSPI algorithm (Algorithm 3) in the toolkit. The choice of GP-SARSA was made upon its dominant performance over the rest statistical PMs [3], whereas LSPI was chosen due to its learning properties and its promising potential in dialogue management when quality features are in use [22].

Both algorithms can be trained either off-line on samples or on-line while the learned policy is used, with very little modifications. In this work we train them off-line in batches, but the learned policies can then be updated online in real-user environments.

We evaluated our algorithms in the following three domains:

(a) *Cambridge Restaurants (CR)*, which is the most common domain in the literature. It produces a 268-dimensional sumBS vector.

(b) *San Francisco Restaurants (SFR)* which has a higher-dimensional (636-dimensional) sumBS vector.

(c) *Laptops11 (LAP11)* which is among the most difficult domains, especially for higher SER. LAP11 has a 257-dimensional sumBS vector.

For each domain we defined a different domain-specific AE, DAE, SDAE or VDAE network. However, the topologies shared exactly the same characteristics for all the domains, besides the size of the input and output layers, which rely on the dimensions of the sumBS vector we want to represent. Furthermore, they shared the same hyperparameters and optimization methods. For instance, we used ADAM optimizer and we applied dropout in the encoder with a rate of 0.6 to avoid overfitting and we run the optimization for 50 epochs during each training.

The learning rate (LR) was configured as time-based, exponential decaying. During pre-training, the initial LR was set to 10^{-1} with a decay rate of 10^{-1} , while for the on-line training the corresponding values were set to 10^{-4} and 10^{-2} 4.1. To

Optimizer	Learning Rate		Decay	Decay Rate		Dropout	Epochs
	Init	Online		Init	Online		
ADAM	10^{-1}	10^{-4}	Exp	10^{-1}	10^{-2}	0.6	50

Table 4.1: Hyper-parameter Values.

obtain fair results, we performed five independent runs of the same experiment using different initialization of the random generator and we calculated average dialogue success rates.

For this work, we consider two distinct BS vectors; the sumBS vector and the DIP vector. DIP as discussed in Section 2.4.3 provides fixed-sized and domain-independent features. Nevertheless, the number of these features is variable and depends on the designer's judgment which of them should be used. For this work we use 58 of them and, thus, the DIP vector has respectively 58 dimensions for every domain. In order to represent the DIP vector using the AE variations, we convert it to its binary form, obtaining a 187-dimensional binary DIP vector.

As far as the topologies are concerned, for the sumBS representation we considered networks for AE and DAE of 5 or 7 fixed-size hidden layers as follows:

- 5-layer: $h1 : 200$, $h2 : 100$, **h3:50**, $h4 : 100$, $h5 : 200$
- 7-layer: $h1 : 200$, $h2 : 100$, $h3 : 50$, **h4:20**, $h5 : 50$, $h6 : 100$, $h7 : 200$

For SDAE we use only the 5-layer architecture and for VDAE only the 7-layer one. When the objective is the representation of the DIP vector, we use a slightly alternated topology of the 5-layer above, where the first and the last hidden layers are smaller:

- 5-layer: $h1 : 150$, $h2 : 100$, **h3:50**, $h4 : 100$, $h5 : 150$

We use the number of the hidden layers in the topology after the name of the AE variation to distinct the different variation-topology combinations. For example, AE5 is the 5-layer topology of the vanilla AE and DAE7 is the 7-layer topology for the DAE and so on.

4.2 Experimental Results

4.2.1 Vanilla and Denoising Autoencoder

Table 4.2 summarizes the performance of the AE5, DAE5 and DAE7 representations against the sumBS one, for the three domains using the GP-SARSA PM. We used four different levels of SER (0%, 15%, 30% and 45%), consistent both in training and evaluation. We run 10 batches of 300 dialogues each and we evaluated each batch on 300 test dialogues. In the environments of 0% SER we do not include the performance of DAEs, as they are identical with a vanilla AE.

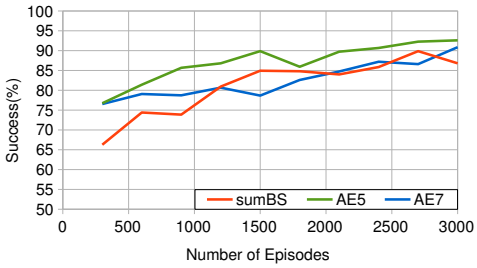
SER	BS	CR	LAP11	SFR
0%	sumBS	98.4%(±1.1)	86.8%(±3.8)	95.2%(±1.3)
	AE5	**99.3%**(±0.5)	**92.6%**(±1.9)	95.3%(±0.8)
	AE7	97.3%(±0.8)	90.9%(±1.7)	**95.4%**(±2.1)
15%	sumBS	96.4%(±2.2)	66.5% (±2.3)	81.6% (±1.6)
	AE5	**96.5%**(±1.0)	68.9%(±10.1)	89.3%(±1.8)
	DAE5	94.5%(±0.7)	80.5%(±0.7)	87.9%(±0.5)
	DAE7	91.9%(±2.9)	**89.7%**(±3.1)	**95.1%**(±1.4)
30%	sumBS	88.5%(±4.2)	51.4%(±9.3)	66.3%(±5.3)
	AE5	92.2%(±1.1)	50.1%(±10.4)	69.4%(±2.3)
	DAE5	92.1%(±0.7)	72.9%(±0.6)	84.0%(±0.8)
	DAE7	**92.9%**(±2.4)	**84.3%**(±4.0)	**94.9%**(±1.28)
45%	sumBS	78.0%(±3.4)	24.1%(±5.5)	53.9%(±6.8)
	AE5	78.1%(±3.3)	38.7%(±5.4)	36.9%(±7.9)
	DAE5	84.2%(±0.8)	76.3%(±0.8)	78.8%(±1.6)
	DAE7	**91.2%**(±5.4)	**88.0%**(±2.9)	**81.9%**(±7.8)

Table 4.2: Average dialogue success after 3000 dialogues using AE and DAE topologies. Standard deviation in parenthesis. Best score in bold.

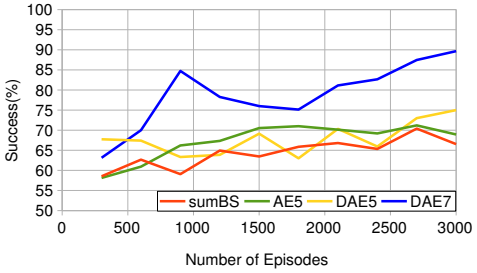
The performance of sumBS with GP-SARSA in our results coincide with those in the benchmarking work in [3], although it slightly differs due to the random initialization of the simulated users. The graphs in Figures 4.1 4.2 and 4.3 show the evolution of the average success rates in the LAP11, CR and SFR domains for different SER levels and the different AE topologies. It can be seen that in all these diagrams, the policy shows a relatively smooth and fast convergence for all the representations. Approximately, the PM needs about 2000 dialogues to achieve convergence.

It can be seen that the dialogue manager benefits from the vanilla AE5 representation for 0% SER, which is consistent for any number of episodes, but is unable to provide noise-robust features for higher SERs. On the other hand, as the presence of noise increases the representation based on the DAE5 and DAE7 topologies show great potential. Their performance is much higher, particularly in the difficult domains of LAP11 and SFR. Specifically, DAE7 maintains a performance close to 90% even for noise levels as high as 45%. Their dominance in performance is even more apparent in the difficult domains of LAP11 and SFR, where the performance of sumBS and vanilla AE5 dramatically degrades. We also observe the dominance of the deeper topologies, as in the extremely noisy environment of 45% SER, the

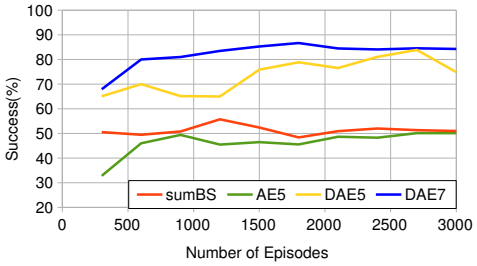
30-dimensional BS representation obtained from DAE7, achieves significantly better performance in the CR and LAP11 domains compared to the 50-dimensional DAE5, and marginally better performance in SFR domain.



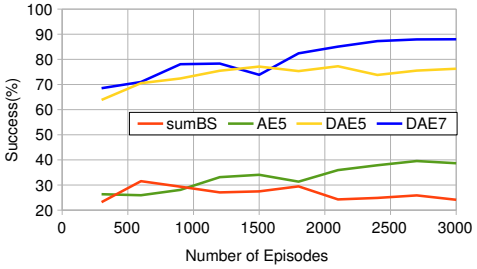
(a) LAP11 0%.



(b) LAP11 15%.

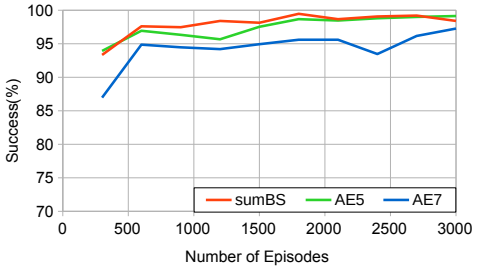


(c) LAP11 30%.

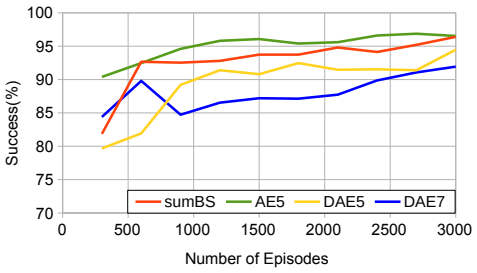


(d) LAP11 45%.

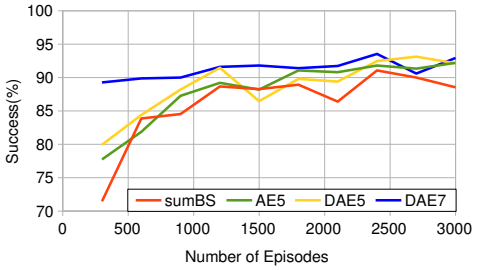
Figure 4.1: LAP11 Domain in Multiple Environments using sumBS and AE/DAE topologies with GP-SARSA for the PM.



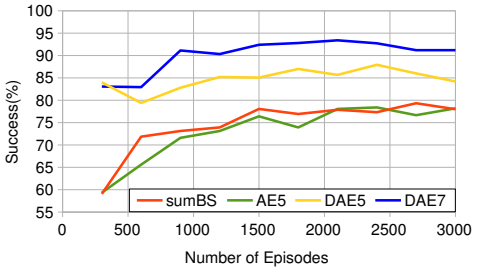
(a) CR 0%.



(b) CR 15%.

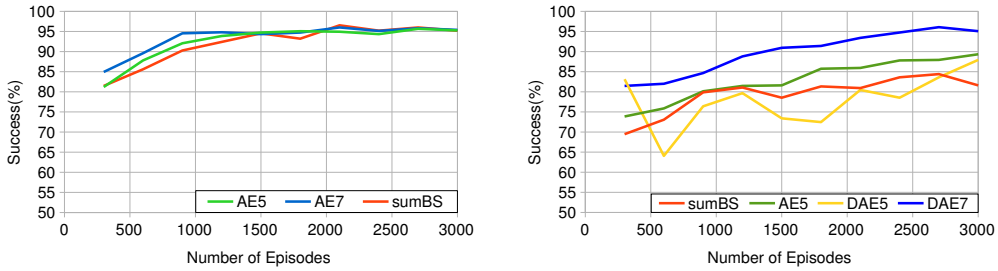


(c) CR 30%.



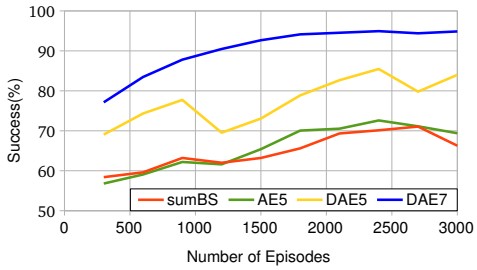
(d) CR 45%.

Figure 4.2: CR Domain in Multiple Environments using sumBS and AE/DAE topologies with GP-SARSA for the PM.

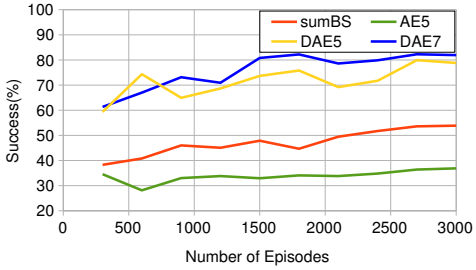


(a) SFR 0%.

(b) SFR 15%.



(c) SFR 30%.



(d) SFR 45%.

Figure 4.3: SFR Domain in Multiple Environments using sumBS and AE/DAE topologies with GP-SARSA for the PM.

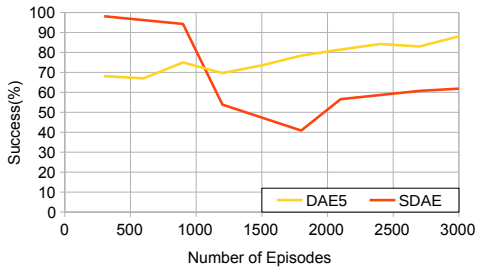
4.2.2 Sparse Denoising Autoencoder

The hypothesis for the utilization of the SDAE representation properties is not quite supported by the PM's performance in different environments, as shown in Figures 4.4, 4.5 and 4.6. We compare the SDAE-based representations with the similar-topology DAE one from the previous section. Observing the diagrams it can be easily inferred that the sparsity in the latent space representation of the sumBS vector prevents the PM from learning a robust policy in any environment. Even thought SDAE provides good representations in the first 300 dialogues and the PM achieves great performance (above 90%), similar to the performance of DAE in 4.2.1, when the sparsity becomes dominant in the latent vector, the performance drops to very low levels. This behavior is consistent in any domain and all the environments. In some cases we even observe better performance of the SDAE in the first 300-600 samples than the DAE5. This is a result of the initialization we performed on the SDAE topology, for which we used the learned parameters of the DAE5 parameters for each domain.

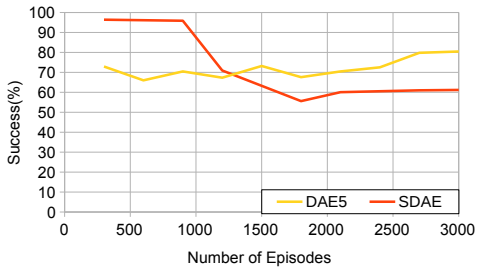
The most probable reason behind the degradation in the performance of the SDAE-based representation is that the sparsity penalty in the loss function, prevents the SDAE to learn informative-enough representations that will lead to a robust policy. In the previous work we mention on 3.4, SDAE's latent space was a lot bigger than the input, a topology that we do not follow in our work. Furthermore, in a SDAE the problem of latent space discontinuity is being intensified through the sparsity parameter, which we brought up in Section 3.5. As a result, the PM fails to learn robust policies to when unseen sumBS vectors provided.

SER	BS	CR	LAP11	SFR
0%	AE5	**99.3%**(±0.5)	**92.6%**(±1.9)	**95.3%**(±0.8)
	SDAE	81.3%(±11.8)	65.0%(±4.1)	77.0%(±7.3)
15%	DAE5	**94.5%**(±0.7)	**80.5%**(±0.7)	**87.9%**(±0.5)
	SDAE	76.3%(±7.3)	70%(±4.5)	70.2%(±7.3)
30%	DAE5	**92.1%**(±0.7)	**72.9%**(±0.6)	**84.0%**(±0.8)
	SDAE	78.4%(±10.1)	64.1%(±9.9)	67.0%(±8.4)
45%	DAE5	**84.2%**(±0.8)	**76.3%**(±0.8)	**78.8%**(±1.6)
	SDAE	75.9%(±11.9)	58.1%(±3.0)	73.3%(±5.1)

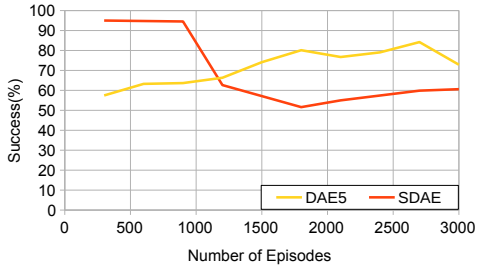
Table 4.3: Average success after 3000 dialogues using DAE5 and SDAE. Standard deviation in parenthesis. Best score in bold.



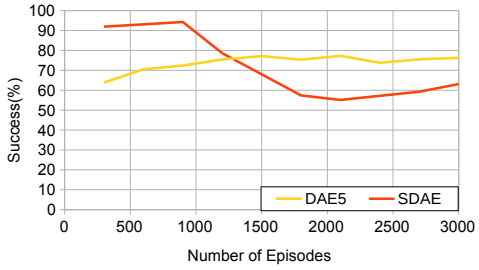
(a) LAP 0%.



(b) LAP 15%.

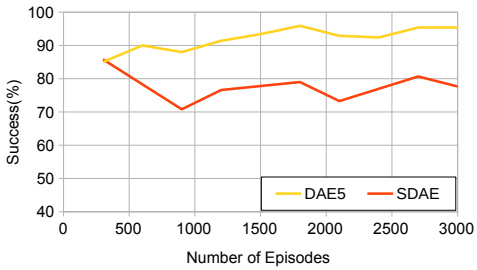


(c) LAP 30%.

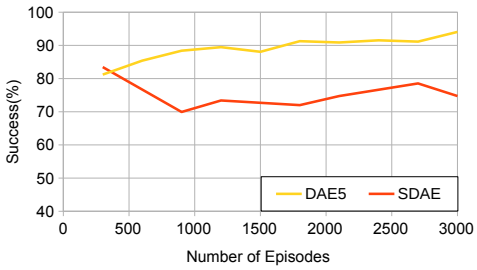


(d) LAP 45%.

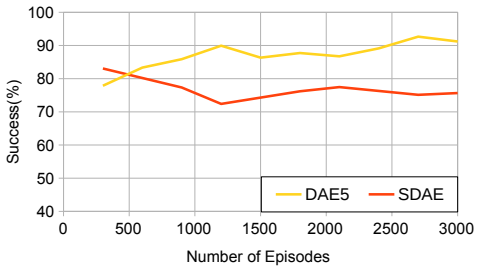
Figure 4.4: LAP Domain in Multiple Environments using DAE5 and SDAE with GP-SARSA for the PM.



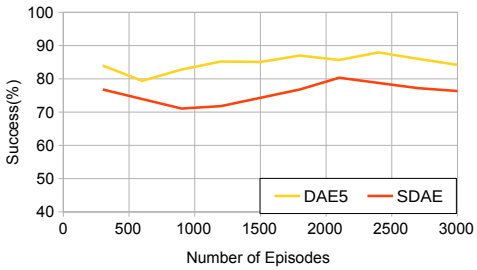
(a) CR 0%.



(b) CR 15%.

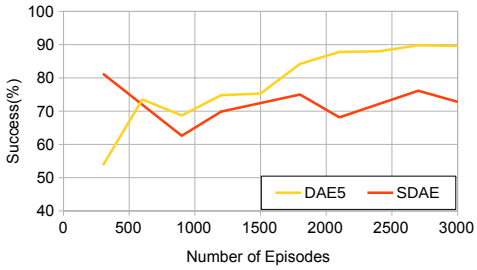


(c) CR 30%.

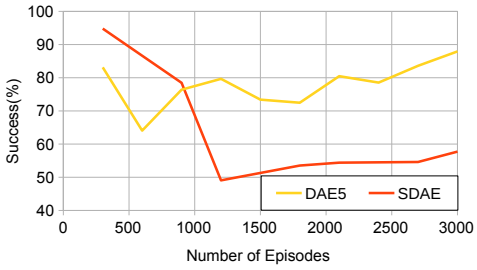


(d) CR 45%.

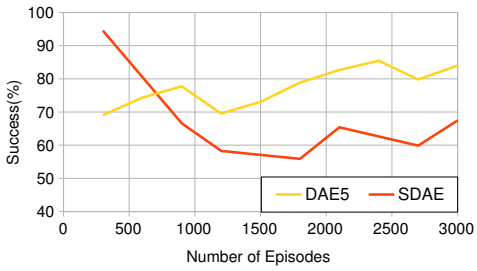
Figure 4.5: CR Domain in Multiple Environments using DAE5 and SDAE with GP-SARSA for the PM.



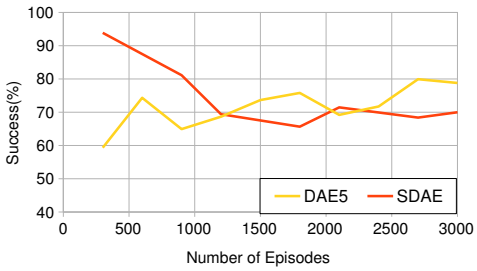
(a) SFR 0%.



(b) SFR 15%.



(c) SFR 30%.



(d) SFR 45%.

Figure 4.6: SFR Domain in Multiple Environments using DAE5 and SDAE with GP-SARSA for the PM.

4.2.3 DIP Features

In order to deal with the deficiencies of DIP stated in 2.4.3 we are representing the DIP vector using DAE in a lower-dimensional space hoping to obtain a more compact and efficient representation of it. Using real-valued DIP features as input to the DAE, caused problems to the backpropagation procedure, as some of the features have wider range of values than others, i.e. feature $1 \in (-3.5, +3.5)$ and feature $2 \in (0, 0.1)$. That gave unintentional importance to the features with wider range and, eventually, prevented DAE to learn good representations. Most of the features was already regularized and we did not want to make any further modifications to them. To overcome this gradient problem we converted the real features into binary ones, obtaining a 187-dimensional binary DIP vector.

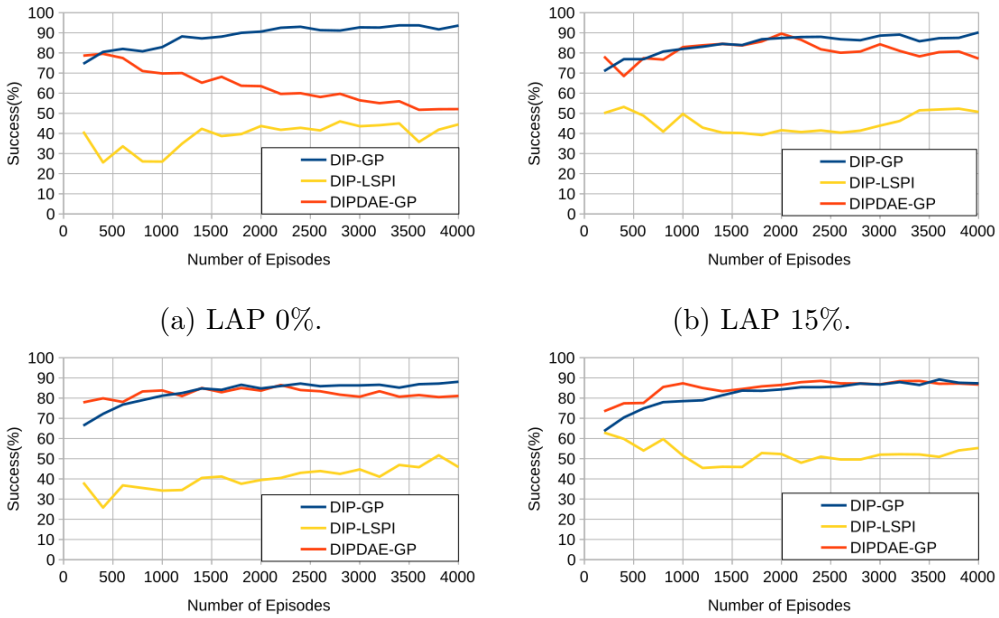
We incorporated the DIP features in the PyDial toolkit and we run experiments with the same training method and set-up as in 3.6 and 4.1 using the GP-SARSA and LSPI algorithms for the PM for 4000 episodes each. The DIP feature vector we are using for these PMs has 58 dimensions each one belonging in $\mathbb{R}$, whereas we use the 187-dimensional binary DIP vector as input to the 5-layer DAE topology as described in 4.1. We call this 50-dimensional BS representation the DIPDAE. The results for the four different environments are shown in table 4.4 and the curves for the three different domains in Figures 4.7 4.8 4.9.

		Domains		
SER	BS	CR	LAP11	SFR
0%	DIP-GP	**97.9%**(±1.38)	**93.6%**(±3.34)	**97.4%**(±1.78)
	DIP-LSPI	44.1%(±27.41)	44.5%(±27.4)	84.4%(±4.2)
	DIPDAE-GP	75.4%(±14.18)	52.1%(±23.48)	60.5%(±14.9)
15%	DIP-GP	**97.2%**(±1.71)	**90.2%**(±5.6)	**91.4%**(±8.1)
	DIP-LSPI	63.7%(±12.6)	50.7%(±28.2)	82.2%(±5.3)
	DIPDAE-GP	76.3%(±4.56)	77.2%(±11.75)	89.1%(±3.27)
30%	DIP-GP	**96.4%**(±1.67)	**88.1%**(±6.62)	**92%**(±4.3)
	DIP-LSPI	60%(±27.6)	45.86%(±25.9)	86.5%(±4)
	DIPDAE-GP	77.4%(±4.36)	**77.4%**(±14.26)	77.6%(±16)
45%	DIP-GP	**96.8%**(±1.56)	87.3%(±2.86)	80.9%(±18.63)
	DIP-LSPI	55.3%(±22.37)	86.7%(±2.88)	72.3%(±17.3)
	DIPDAE-GP	68.4%(±16)	**88%**(±2.8)	**86.3%**(±7.2)

Table 4.4: Average dialogue success after 4000 dialogues using DIP. Standard deviation in parenthesis. Best score in bold.

The use of DIP features enables PM with GP-SARSA to learn good and relatively stable policies across the three domains in every environment 4.4. The observed performance of our experiments is again consistent with the performance shown in

[37]. For CR domain the success rate is very high and the rise of SER in the 4 environments does not affect PM. However, in LAP11 and SFR domains the drop in the success rate reaches a decrease of 17% from 0% to 45% SER. Overall, it is apparent that DIP can provide some tolerance in noise, by using only domain-independent information from SLU module (Section 2.1).



(a) LAP 0%.

(b) LAP 15%.

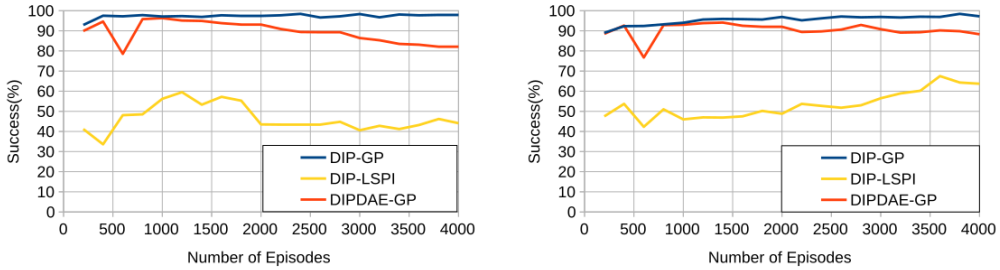
(c) LAP 30%.

(d) LAP 45%.

Figure 4.7: LAP Domain in Multiple Environments using DIP with GP-SARSA and LSPI for the PM.

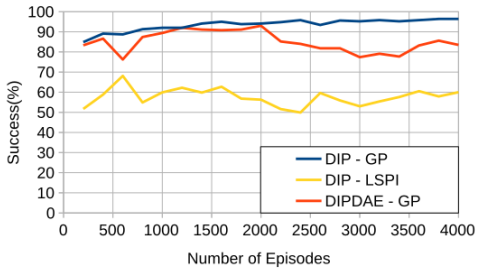
An interesting observation is that DIP features did not performed well when used in combination with the LSPI algorithm, as they lead to inadequate policies during dialogues in domains LAP11 and CR. However, they seem to help LSPI to learn good policies in SFR domain, but yet the performance is not comparable to the GP-SARSA PM's one. The reason of the exhibited performance can be traced to the basis function (Section 2.3.2) we use at LSPI, which correlates at a high degree the BS representations, obtained from DIP, with the function approximation. If the DIP features are not compact or informative enough, LSPI will not be able to learn a good approximation of the Q-function and, consequently, a good policy.

Another indication that DIP features are not informative enough is that DAE5 cannot find a robust representation in some experiments. This is shown in Figures 4.7, 4.9 and 4.8, where the success rate of the PM with GP-SARSA is dropping throughout the training of the PM and the dynamic adjustment of the DAE5.

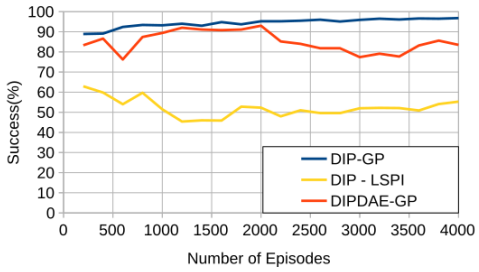


(a) CR 0%.

(b) CR 15%.

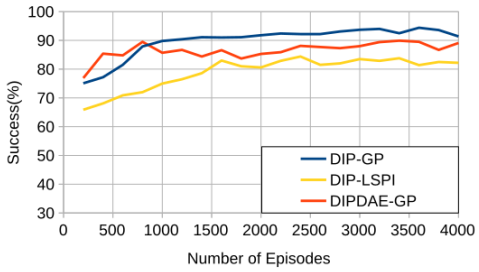


(c) CR 30%.

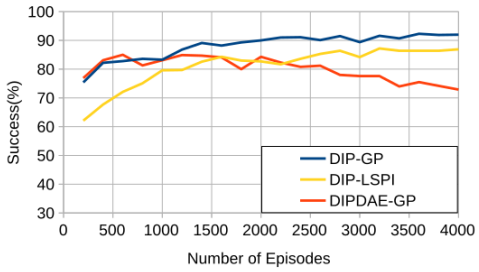


(d) CR 45%.

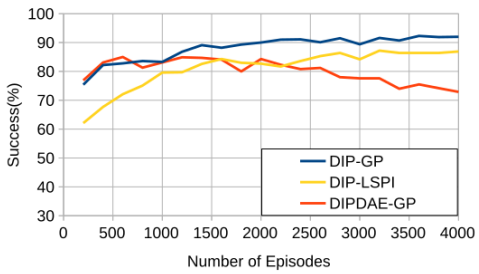
Figure 4.8: CR Domain in Multiple Environments using DIP with GP-SARSA and LSPI for the PM.



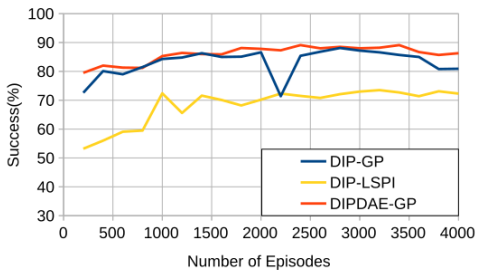
(a) SFR 0%.



(b) SFR 15%.



(c) SFR 30%.



(d) SFR 45%.

Figure 4.9: SFR Domain in Multiple Environments using DIP with GP-SARSA and LSPI for the PM.

4.2.4 Variational Denoising Autoencoder

The outstanding performance of the DAE-based representations from the deeper 7-layer topology of the DAE (Section 4.2.1) motivated us to focus our experiments only on deeper topologies for the VDAE. We also used a 5-layer VDAE to represent the DIP feature vector. The reason why we use a more swallow topology, was to directly compare the DIPVDAE representation with the DIPDAE one. Based on the approach in 4.2.3, we gave the 187-dimensional binary DIP vector as input to the VDAE.

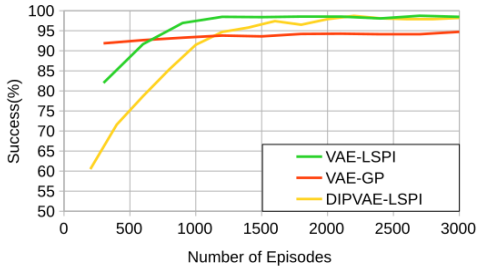
For the experiments in 0% SER the VDAE is in fact a VAE, as the sumBS vector we give as input is noise-free. We again run the experiments in 10 batches of 300 dialogues each, since we did not observed any improvement in the PM's behavior in the previous section after 3000 dialogues.

Samples: 3000		**Domains**		
SER	BS	CR	LAP11	SFR
0%	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)
	DIPVAE-LSPI	80.7%(±9.0)	**99.1%**(±0.7)	95.0%(±1.8)
15%	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)
	DIPVDAE-LSPI	80.4%(±1.9)	97.4%(±2.3)	94.7%(±2.6)
30%	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)
	DIPVDAE-LSPI	81.8%(±2.3)	97.0%(±1.8)	92.8%(±2.41)
45%	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)
	DIPVDAE-LSPI	79.1%(±4.0)	**96.7%**(±2.6)	92.4%(±2.5)

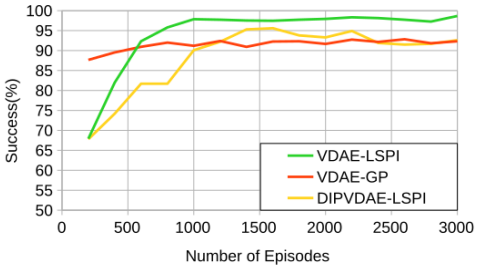
Table 4.5: Average dialogue success after 3000 dialogues using VDAE and DIPVDAE. Standard deviation in parenthesis. Best score in bold.

The performance of the VDAE-based representations is prominent in Table 4.5. The PM that utilizes LSPI to learn the dialogue policies show the true potential of the algorithm when compact and highly informative representations of the sumBS vector are provided. The average dialogue success rate of the LSPI-based PM is astonishingly high for the 3 domains in every environment. Even though there is an average decrease of 4% from 0% SER to 45% SER, still the success rate is above 93.6%. Additionally, VDAE shows very quick convergence, from the first 300 sampels with GP-SARSA and under 1000 samples with LSPI (Figures 4.12. 4.10 and 4.11). T

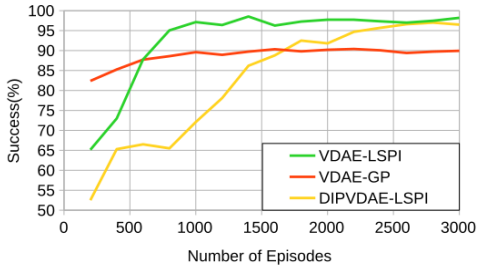
The quality of VDAE representations is also apparent when used in combination with the GP-SARSA algorithm, yet this PM never outperforms the LSPI-based one.



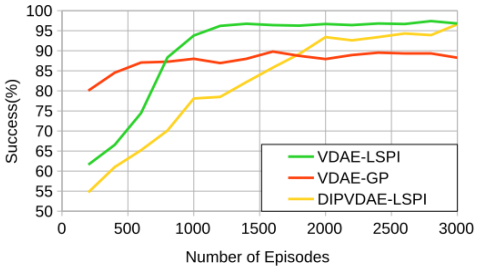
(a) LAP 0%.



(b) LAP 15%.

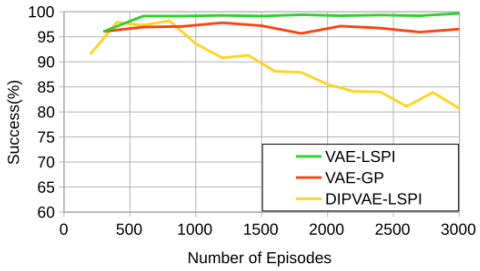


(c) LAP 30%.

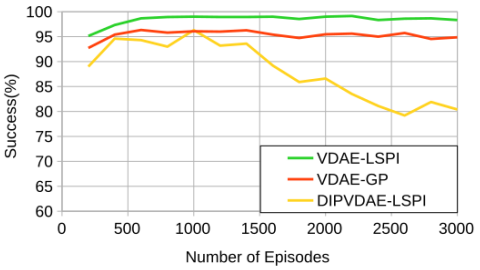


(d) LAP 45%.

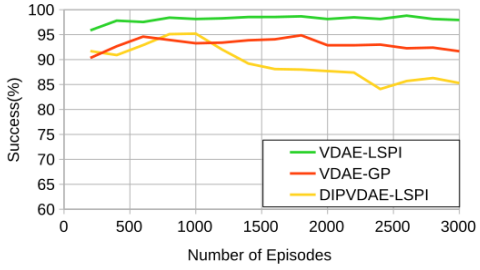
Figure 4.10: LAP Domain in Multiple Environments using VAEVDAE and DIP-VAE/DIPVDAE with GP-SARSA and LSPI for the PM.



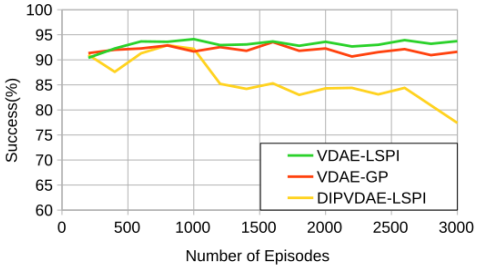
(a) CR 0%.



(b) CR 15%.



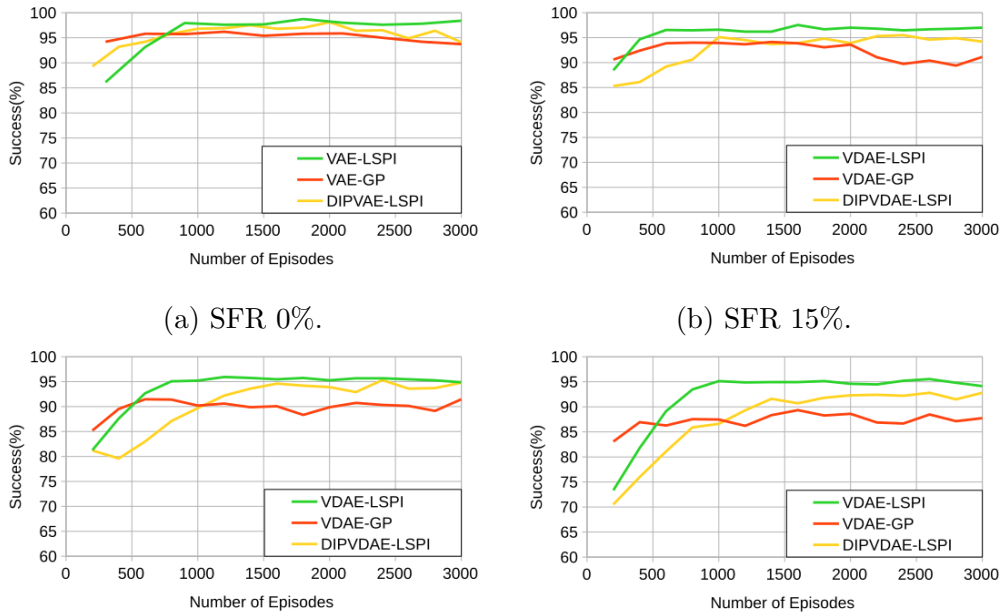
(c) CR 30%.



(d) CR 45%.

Figure 4.11: CR Domain in Multiple Environments using VAEVDAE and DIP-VAE/DIPVDAE with GP-SARSA and LSPI for the PM.

The performance of the VDAE in combination with GP-SARSA is equivalent to the DAE7-based representations 4.2, as well as to the PM that uses the DIP features (Section 4.4). These diagrams demonstrate the consistent behaviour of the VDAE among all the domains since in all cases it achieves high accuracy with the smallest



(a) SFR 0%.

(b) SFR 15%.

(c) SFR 30%.

(d) SFR 45%.

Figure 4.12: SFR Domain in Multiple Environments using VAEVDAE and DIP-VAE/DIPVDAE with GP-SARSA and LSPI for the PM.

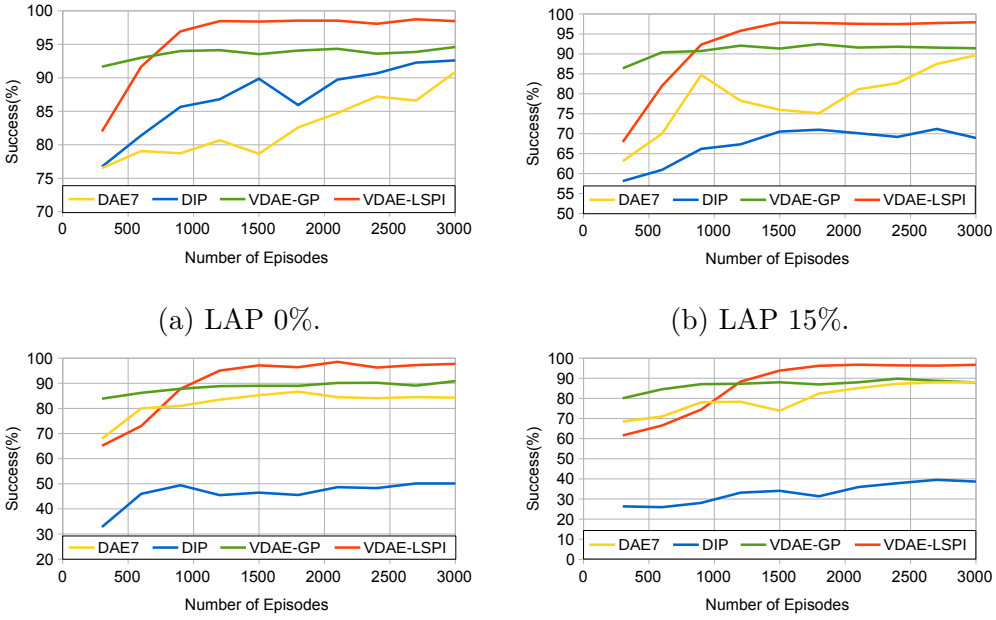
number of training episodes.

The representations of the DIP features we get from the VDAE exhibit unreliable performance. Even though in CR domain they fail to give robust representations to the PM, in LAP11 and SFR the DIPVDAE representations enable PM to reach performances similar to when we use the VDAE representation of the sumBS vector. DIPVDAE representation during CR domain simulations seems to have stuck in a local minimum during training, which gave this bad performance. This behavio is consistent to the one of the DIPDAE representations (Figures 4.7, 4.8 and 4.9). Generally, we conclude that AE variations show instability when try to learn representations of the DIP features and more attention during their training should be paid to ensure quality representations.

4.3 Discussion

In Figures 4.13, 4.14 and 4.15 we include the performance curves of DIP features with GP-SARSA and the most dominant AE-based representations of the sumBS space; DAE7 with GP-SARSA, VDAE with GP-SARSA (VDAE-GP) and VDAE with LSPI (VDAE-LSPI). Thus, we can easily compare their performance and distinguish the most prominent one. The combination of VDAE with LSPI (red curve) clearly outperforms every other BS representation and PM combination. VDAE representations of the sumBS space converge faster than any other BS representation, with this behavior being consistent for every PM, domain or SER level. On the contrary, VDAE in combination with GP-SARSA, even though it consistently

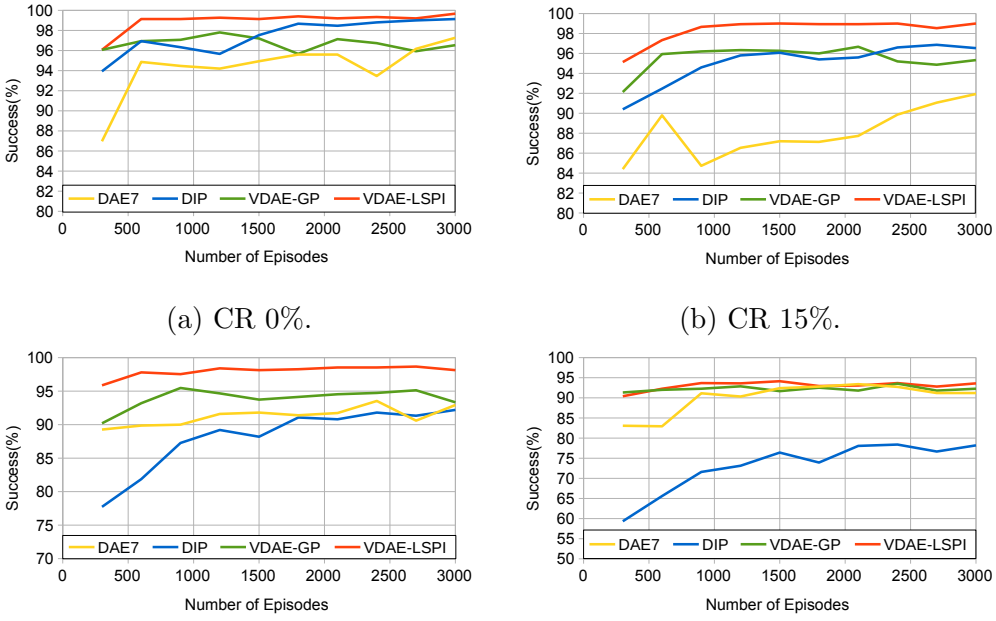
maintains higher performance than DAE7 and DIP for LAP11 domain, for CR domain it is outperformed by DIP in 0% and 15% SER environments and for SFR domain it is outperformed by DAE7 for every SER level, except the 45% one.



(a) LAP 0%. (b) LAP 15%.

(c) LAP 30%. (d) LAP 45%.

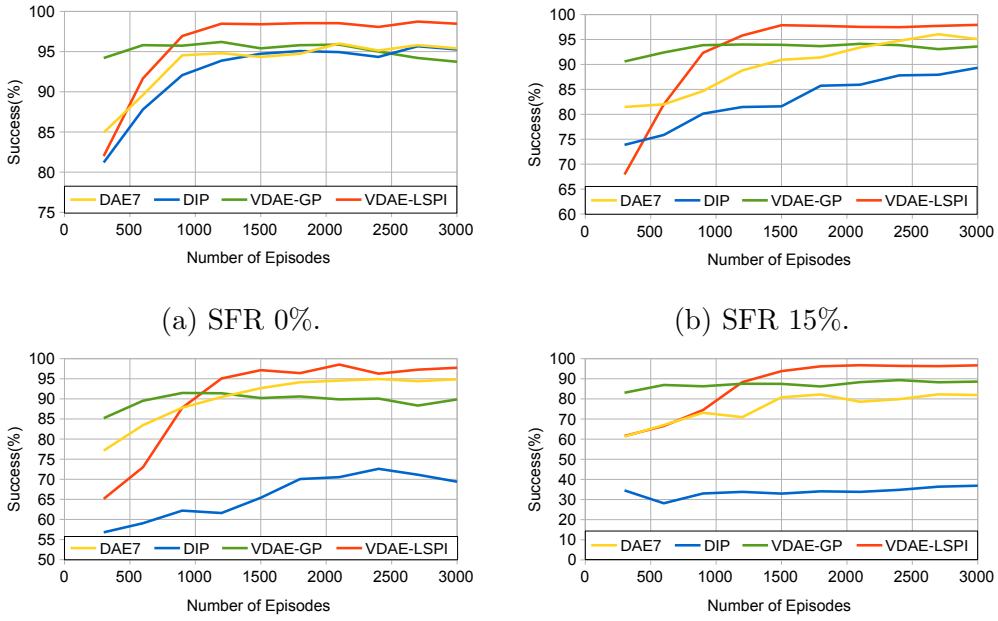
Figure 4.13: Comparison between DAE7, DIP and VDAE with GP-SARSA and VDAE with LSPI for the LAP Domain in Multiple Environments.



(a) CR 0%. (b) CR 15%.

(c) CR 30%. (d) CR 45%.

Figure 4.14: Comparison between DAE7, DIP and VDAE with GP-SARSA and VDAE with LSPI for the CR Domain in Multiple Environments.



(a) SFR 0%.

(b) SFR 15%.

(c) SFR 30%.

(d) SFR 45%.

Figure 4.15: Comparison between DAE7, DIP and VDAE with GP-SARSA and VDAE with LSPI for the SFR Domain in Multiple Environments.

A direct comparison with the state-of-the-art is not feasible due to the different experimental protocols, domains, and policies of previous works. For example, in [37], a first version of the DIPs in conjunction with GP-SARSA is tested on both the CR and a precursor of the LAP11 domain. After training on 1000 dialogues with 15% SER, the reported accuracy is approximately 95% for the CR, which is comparable to the average accuracy of 94.6% that we obtain using dense AEs and lower than the accuracy of 96.2% of the VDAE in the same environment and domain after 900 dialogues. In [18], where the CR domain is studied, the examined BS representation is DIPs, BinLin, and BinAux and the policy manager is A2C, whereas the number of training dialogues is 2000. The reported accuracy ranges from 17% to 35% lower compared to our VDAE system, depending on the environment.

In [4, 5] DIP, FFN and RNN Feudal features are exploited. The policies considered are NN-based and the results presented are for 4000 episodes. Nevertheless, since they consider the same domains and matching conditions for the environments 1, 3 and 6 which correspond to 0%, 15% and 30% respectively, we show in Table 4.6 a comparison of both the average success and the reward. It can be seen that the proposed VDAE with LSPI (VDAE-LSPI) dominates in all cases in terms of success rate, as well as in terms of average reward, except the SFR domain, where DAE7 with GP-SARSA (DIP7-GP) obtains the highest one. In addition, this performance gain achieved despite the fact that VDAE-LSPI and DAE7-GP where trained on less episodes.

As far as the SDAE representations of the sumBS vector are concerned, the SDAE topology is unable to compress the information of the sumBS vector into a low-dimension one, while imposing sparsity to it. The applications of the SAE in the

SER/ ENV	Model	CR		LAP11		SFR	
		Succ	Rew	Succ	Rew	Succ	Rew
0%	DAE7-GP	97.3%	13.1	90.9%	10.6	95.4%	**11.9**
	VDAE-LSPI	**99.7%**	**13.3**	**98.5%**	**11.4**	**98.4%**	11.3
Env.1	Feudal-DQN	89.3%	11.7	65.5%	5.7	71.1%	7.1
15%	DAE7-GP	91.9%	11.7	89.7%	10.1	95.1%	**11.5**
	VDAE-LSPI	**99.0%**	**13.1**	**97.9%**	**11.2**	**97.0%**	10.9
Env.3	Feudal-DQN	92.6%	11.7	89.6%	9.4	90.0%	9.7
30%	DAE7-GP	93.0%	11.9	84.3%	9.0	94.9%	**11.2**
	VDAE-LSPI	**98.1%**	**12.7**	**97.7%**	**10.8**	**95.3%**	9.8
Env.6	Feudal-DQN	90.6%	10.4	78.5%	6.0	83.0%	7.1

Table 4.6: DAE7-GP and VDAE-LSPI average success and reward (3000 episodes) compared to the Feudal-DQN system [5] (4000 episodes)

literature ([25], [44] and [43]), use a higher-dimensional latent space in comparison with the input of the encoder, in order to learn a sparse representation of the input. This difference in our approach is probably the reason of SDAE's bad performance. However, we cannot make use of such a representation, since it would be harder for the PM to learn good policies based on that. In addition, in the aforementioned cases where the SAE is used with success, the objective is classification task and not a control problem.

It is also worth noting that since the all the AE topologies in our proposed scheme is domain-independent with the exception of the input and output layer, new AEs could be rapidly optimized in a transfer learning framework [14, 46]. This is particularly useful when new entries in a domain's ontology are introduced, as well as for optimizing new AEs for new domains with limited data, making our approach domain-transferable.

Chapter 5

Conclusion

In this thesis, we are proposing a novel use of of the vanilla AE, DAE, and VDAE for representing the sumBS space for SDMs. Our motivation is to create low-dimensional, fixed-size, and robust representations which are automatically extracted from a set of training dialogues. Using artificially corrupted training data we obtain representations tolerant in SER noise. Furthermore, the learned representations are trained concurrently with the PM in a novel training scheme. The experimental results show that the proposed scheme is very efficient, outperforming significantly, even with limited training samples, the baseline sumBS, and other state-of-the-art approaches for different domains and noise levels. The proposed representations were tested in different domains, namely Cambridge Restaurants, San Francisco Restaurants and Laptops11 under several SER conditions ranging from 0% to 45%. The novelty and efficiency of the use of the DAE and VDAE is confirmed through the constantly high performance under all conditions.

In Section 4.2.1 we show that a DAE topologies can learn representations in a low-dimansional latent space that are also noise-tolerant. In the same section we observed a promising behavior of a deeper architecture with lower-dimensionality of the latent space. In Section 4.2.2 our initial hypothesis for the capabilities of a SDAE to provide compact representations, while also enforcing the latent space representation to provide a sparse vector, was not supported by the results of our experiments. We observed that while the representations of SDAE were becoming sparser, the performance of the PM was dropping constantly.

In this thesis, we also explored the potential of DIP features in 4.2.3 to provide domain-independent features, from which we can eventually extract lower-dimensional and noise-robust representations for use in the PM. One the one hand, DIP features exhibited very bad performance when used in combination with the LSPI algorithm. On the other hand, DAE and VDAE 5-layer topologies could not learn a consistently efficient representation of DIP features. Thus, we conclude that DIP features introduce instability during the training of the two variations of AEs and deeper insight in their optimization should be gained.

Nevertheless, VDAE representations of the sumBS space exhibited outstanding performance, as it is shown on our experiments in Section 4.2.4. VDAE showed the great potential of NN-based representations in SDSs. The orientation of our

efforts to learn the distribution that produces the latent space representations of the sumBS space, instead the representation itself, proved to be fruitful. The PM, regardless the RL algorithm in use, managed to learn exceptional and noise-tolerant policies. Apparently, LSPI consistently outperformed even the most popular and robust GP-SARSA algorithm for every domain and in every environment with the use of VDAE-based representations.

Consequently this work, proposes a novel use of different variations of the AE as an automated representation technique for low-dimensional and robust representations of the sumBS space. We started exploring its most simple variations of the AE (vanilla AE) and proceeded in the more sophisticated ones (DAE and VDAE) performing eventually variational inference at the latent space. In addition we tried to achieve domain-independence representing the DIP features, even though we did not managed to reach the desired performance we were seeking. Finally, we tested the acquired representations on a PM that utilized two state-of-the-art RL algorithms; the non-parametric GP-SARSA and the parametric LSPI.

The experimental results show that the proposed scheme is very efficient, outperforming significantly, even with limited training samples, the baseline sumBS, and other state-of-the-art approaches for different domains and noise levels. The highest gain refers to environments with 45% SER and difficult domains such as the demanding Laptops11, where an absolute improvement of 72.6% is achieved (sumBS with GP-SARSA vs. VDAE with LSPI), confirming the proposed method's ability.

Future work includes the use of VDAE representations with non-linear PM, like DQN, eNAC or A2C, in order to explore their potential with another popular category of RL algorithms for SDM systems.

Acknowledgements

This thesis was part of the collaborative research work between the Telecommunications Systems Institute (TSI) of Technical University of Crete (TUC) and the Cambridge Research Laboratory (CRL) of Toshiba Research Europe Limited (Toshiba). DIP is a tool developed in CRL and is commercially patented from Toshiba and all the experiments that acquired its use were held exclusively in CRL, as disclosure agreements are in use. This is the reason why we did not proceeded in examining more thoroughly the behavior of the DAE and VDAE when used to represent DIP features, as the presence in CRL was limited in time.

Big part of this work has been submitted at the ASRU 2019 conference with title "Robust Belief State Space Representation for Statistical Dialogue Managers using Deep Autoencoders" and it includes the experiments on the AE, DAE and VDAE representations of the sumBS the summary space.

Bibliography

[1] Richard Bellman. "A Markovian Decision Process". In: *Indiana Univ. Math. J.* 6 (4 1957), pp. 679–684. ISSN: 0022-2518.

[2] Pawel Budzianowski et al. "Sub-domain Modelling for Dialogue Management with Hierarchical Reinforcement Learning". In: *SIGDIAL*. 2017.

[3] Iñigo Casanueva et al. "A Benchmarking Environment for Reinforcement Learning Based Task Oriented Dialogue Management". In: *CoRR* abs/1711.11023 (2017).

[4] Iñigo Casanueva et al. "Feudal Dialogue Management with Jointly Learned Feature Extractors". In: *SIGDIAL*. 2018.

[5] Iñigo Casanueva et al. "Feudal Reinforcement Learning for Dialogue Management in Large Domains". In: *NAACL-HLT*. 2018, "714–719".

[6] Mehdi Fatemi et al. "Policy Networks with Two-Stage Training for Dialogue Systems". In: *SIGDIAL*. 2016.

[7] Milica Gasic and Steve J. Young. "Gaussian Processes for POMDP-Based Dialogue Manager Optimization". In: *IEEE/ACM Trans. Audio, Speech, and Language Processing* 22 (2014), pp. 28–40.

[8] Kuan Han et al. "Variational Autoencoder: An Unsupervised Model for Modeling and Decoding fMRI Activity in Visual Cortex". In: *bioRxiv* (2018).

[9] Hilda Hardy, Tomek Strzalkowski, and Min Wu. "Dialogue Management For An Automated Multilingual Call Center". In: *HLT-NAACL*. 2003.

[10] 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.

[11] Ridong Jiang et al. "Component Plugguble Framework and Its Application to Social Robots". In: *Natural Interacton with Robots and Smartphones, Putting Spoken Dialog Systems into Practice*. 2014.

[12] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and Acting in Partially Observable Stochastic Domains". In: *Artif. Intell.* 101 (1998), pp. 99–134.

[13] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101 (1998), pp. 99–134.

[14] C. Kandaswamy et al. "Improving transfer learning accuracy by reusing Stacked Denoising Autoencoders". In: *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2014, pp. 1380–1387.

[15] Kyungduk Kim et al. "A Frame-Based Probabilistic Framework for Spoken Dialog Management Using Dialog Examples". In: *SIGDIAL*. 2008.

[16] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *CoRR* (2014).

[17] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[18] Margarita Kotti et al. "A Case Study on the Importance of Belief State Representation for Dialogue Policy Management". In: *INTERSPEECH* 986-990 (2018).

[19] Alex Krizhevsky and Geoffrey E Hinton. "Using very deep autoencoders for content-based image retrieval." In: *ESANN*. 2011.

[20] Michail G. Lagoudakis and Ronald Parr. "Least-Squares Policy Iteration". In: *Journal of Machine Learning Research* 4 (2003), pp. 1107–1149.

[21] Yann Lecun. "A theoretical framework for back-propagation". In: *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*. Ed. by D. Touretzky, G. Hinton, and T. Sejnowski. 1988.

[22] Lihong Li, Jason D. Williams, and Suhrid Balakrishnan. "Reinforcement learning for dialog management using least-squares Policy iteration and fast feature selection". In: *INTERSPEECH*. 2009.

[23] De-Rong Liu, Hong-Liang Li, and LiDing Wang. "Feature Selection and Feature Learning for High-dimensional Batch Reinforcement Learning: A Survey". In: *International Journal of Automation and Computing* 12 (2015), pp. 229–242.

[24] Xugang Lu et al. "Speech enhancement based on deep denoising autoencoder." In: *INTERSPEECH*. 2013, pp. 436–440.

[25] Alireza Makhzani and Brendan J. Frey. "k-Sparse Autoencoders". In: *CoRR* abs/1312.5663 (2014).

[26] Håkan Melin, Anna C Sandell, and Magnus Ihse. "CTT-bank: A speech controlled telephone banking system - an initial evaluation". In: *Speech Music and Hearing Quarterly Progress and Status Report (TMH-QPSR)*. Vol. 1. 2001.

[27] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013).

[28] Alexandros Papangelis et al. "An adaptive dialogue system for assessing post traumatic stress disorder". In: *PETRA*. 2013.

[29] Harris Partaourides and Sotirios P. Chatzis. "Asymmetric Deep Generative Models". In: *Neurocomputing* 241 (2017), pp. 90–96.

[30] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22 (1951), pp. 400–407.

[31] Nicholas Roy, Joellea Pineau, and Sebastian Thrun. "Spoken Dialogue Management Using Probabilistic Reasoning". In: *ACL*. 2000.

[32] ALICE E. SMITH and ANTHONY K. MASON. "COST ESTIMATION PREDICTIVE MODELING: REGRESSION VERSUS NEURAL NETWORK". In: *The Engineering Economist* (1997), pp. 137–161.

[33] Pei-Hao Su et al. "Sample-efficient Actor-Critic Reinforcement Learning with Supervised Data for Dialogue Management". In: *SIGDIAL*. 2017.

[34] "Two-Point Step Size Gradient Methods". In: *IMA Journal of Numerical Analysis* (1988), pp. 141–148.

[35] Stefan Ultes et al. "PyDial: A Multi-domain Statistical Dialogue System Toolkit". In: *ACL*. 2017.

[36] Pascal Vincent et al. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *J. Mach. Learn.Res* (2010), pp. 3371–3408.

[37] Zhuoran Wang et al. "Learning Domain-Independent Dialogue Policies via Ontology Parameterisation". In: *SIGDIAL*. 2015.

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

[39] Jason Williams and Steve Young. "Scaling POMDPs for spoken dialog management". In: *Audio, Speech, and Language Processing, IEEE Transactions on* 15 (2007), pp. 2116–2129.

[40] Steve J. Young. "Talking to machines (statistically speaking)". In: *INTERSPEECH*. 2002.

[41] Steve J. Young et al. "POMDP-Based Statistical Spoken Dialog Systems: A Review". In: *Proc. IEEE* 101 (2013), pp. 1160–1179.

[42] S. Young et al. "The Hidden Information State Approach to Dialog Management". In: *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*. Vol. 4. 2007.

[43] Nianyin Zeng et al. "Facial expression recognition via learning deep sparse autoencoders". In: *Neurocomputing* 273 (2018), pp. 643–649.

[44] Changfan Zhang et al. "Deep Sparse Autoencoder for Feature Extraction and Diagnosis of Locomotive Adhesion Status". In: 2018.

[45] G. P. Zhang. "Neural networks for classification: a survey". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* (2000), pp. 451–462.

[46] Fuzhen Zhuang et al. "Supervised Representation Learning: Transfer Learning with Deep Autoencoders". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. 2015, pp. 4119–4125.