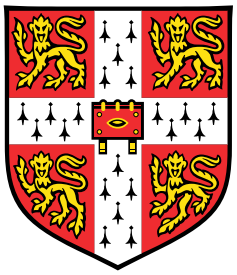
Building a Conversational User Simulator Using Generative Adversarial Networks



Caroline Dockes

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Trinity College August 2021

Declaration

I, Caroline Dockes of Trinity College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. The report makes use of the following software:

- The PyTorch machine learning library, for building and training the various models discussed. See https://pytorch.org/.
- The NumPy (https://numpy.org/), matplotlib (https://matplotlib.org/), pandas (https://pandas.pydata.org/), and scikit-learn (https://scikit-learn.org/) libraries for data wrangling and plotting tasks.
- The Plato Research Dialogue System v0.3.1 for training dialogue policies. See https://github.com/uber-archive/plato-research-dialogue-system.
- Toshiba's own dialogue system training set-up, which is built using the Plato platform.

Word count: 13,690

Caroline Dockes
August 2021

Abstract

User simulators are valuable tools for training task-oriented dialogue systems. In past work, they have generally either been based on hand-crafted rules or trained using maximum likelihood estimation (MLE). In this dissertation, we build the first such simulator based on a generative adversarial network (GAN). The proposed model uses a long short-term memory encoder-decoder architecture. It takes in structured inputs representing dialogue context, and outputs sequences of user dialogue acts.

We carry out a number of experiments with the goal of improving the adversarial training process. We then compare the resulting model with an MLE trained simulator. We measure the simulators' performances directly, based on metrics such as the F-score, and indirectly, by using them to train dialogue system policies via reinforcement learning, and evaluating these policies against several different simulators. Our results suggest that, even though the GAN simulator obtains worse results on direct metrics, it trains better policies than the MLE simulator.

Acknowledgements

I would first like to thank my two supervisors, Dr. Simon Keizer and Dr. Kate Knill. They have provided me with helpful suggestions and feedback, as well as consistent support throughout the project. I am grateful for their time and input. I would also like to thank Dr. Svetlana Stoyanchev and Dr. Rama Doddipatla at Toshiba for helpful discussions. Furthermore, I want to acknowledge Toshiba for providing me with computing resources, and Trinity College for financial support. Above all, I am grateful for the support of my friends, my siblings, and my parents.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

Task-oriented dialogue systems are programs that interact with human users through natural language to achieve specific goals, such as booking a flight or setting an alarm. Well known examples of such systems include Amazon's Alexa, Apple's Siri and Google's assistant.

State-of-the-art systems from the literature are usually trained using reinforcement learning (RL) methods. This process generally requires a large number of interactions between the system and a user. To avoid the high cost of involving humans in development, a common approach is to replace the user with a simulator during training. The simulator is a model that interacts with the dialogue system, making requests and communicating in the same way that a human user would.

For such training to translate into good performance against real users, the simulator must closely imitate human behaviour. It must act rationally, but also produce diverse behaviours that will prepare the dialogue system for a wide variety of situations. This is challenging, and a large literature exists which tries to improve such simulators. A range of approaches have been proposed, from rule-based to neural models. To the best of our knowledge however, no work has looked at using a generative adversarial network (GAN) for this purpose.

GANs are intuitively appealing for this task as their training objective aligns well with the goal of user simulation. Indeed, they are trained to fool a discriminator model into thinking that their generated output is genuine data from human users, essentially learning to pass an automated Turing test. Of course, since the discriminator's only reference for human behaviour is the training corpus, we do not expect GANs to be able to stray too far from the data in their output. Even so, we hypothesise that, relative to traditional supervised learning approaches, adversarial training will put less emphasis on reproducing the training examples exactly and may produce more useful simulators.

Also encouraging is the fact that GANs have shown impressive performance on other data generation tasks. Admittedly, many of these applications revolved around image generation, and GANs have generally performed worse in the context of sequence generation, particularly text generation. We do not think that this rules them out as promising approaches to user simulation for two reasons. First, recent papers have investigated ways of stabilising and improving adversarial training for discrete outputs, and we take these findings into account when developing our own model. Second, the task of user simulation is distinct from text generation. In particular, our own model will focus on generating sequences of dialogue acts, producing semantic content rather than natural language.

1.2 Contributions

Our most important contribution is building the first task-oriented dialogue system user simulator based on a GAN architecture. Our results suggest that the proposed model trains better dialogue systems than a baseline simulator trained with maximum likelihood estimation (MLE), encouraging further investigation into this use for GANs.

We also make two secondary contributions. First, we add to the body of work that investigates how adversarial training can be improved for sequence generation by describing how various aspects of the training process affect our model's performance.

Second, our results are interesting for the problem of simulator evaluation. We provide evidence of a discrepancy between direct evaluation metrics such as the F-score, and indirect evaluation methods which measure a simulator's quality by the performance of dialogue policies trained against it. Indeed, we show that while the GAN simulator is outperformed by an MLE baseline on the former, it obtains better results on the latter. We will argue that indirect evaluation is more relevant to the actual goal of user simulation, and should be privileged when available.

1.3 Outline

The dissertation is organised into seven chapters, starting with this introduction. The next chapter reviews relevant literature around user simulation and adversarial training, and provides theoretical background relating to GANs. Chapter 3 describes our proposed model. Chapter 4 provides details on our experimental methods, including a description of the training corpus. In chapter 5, we discuss and justify the various choices that have been made with respect to the adversarial training process. Chapter 6 presents our main results, which compare our proposed simulator with an MLE trained baseline. We conclude in chapter 7

by summarising our contributions, exploring their limitations and making suggestions as to further work.

Chapter 2

Background

This section provides context for this dissertation, including an overview of the relevant literature. We first describe the problem of user simulation for task oriented dialogue systems and discuss how simulators are evaluated. We then introduce GANs and describe past work employing them for sequence generation.

2.1 User Simulation for Task-Oriented Dialog Systems

2.1.1 Why Do We Need User Simulation?

The purpose of a task-oriented dialogue system is to assist users in accomplishing specific tasks. The system interacts with users through natural language to identify what it is they want to achieve. In the relatively simple example of a restaurant booking system, the system must interact with the user to determine their preferences, for example in terms of cuisine or geographical area, and can then suggest restaurants, and provide the user with information. In the process, the system can ask for clarifications, or can suggest that the user change their constraints if no relevant option is available. Therefore, even for such a simple goal, the system's task is complex, and its success is only determined at the end of the interaction. This is why RL methods are generally used to train such systems (Dhingra et al., 2017; Peng et al., 2017; Su et al., 2016; Young et al., 2013; Zhou et al., 2017).

Unfortunately, training a policy with RL necessitates that the system have many interactions, which is costly to do with human users. One common solution is to employ a user simulator during training. A corpus of existing dialogues is still needed to train the simulator, but once this is done, the dialogue system can have a large number of interactions with it. The system is also able to explore the policy space in a way that would not be possible using a static corpus. Indeed, in the corpus, the system (or sometimes human operator)

interacting with the user is itself following a policy, and that policy may not be optimal. Using a simulator allows the RL training process to explore strategies not used in the corpus[1].

We now describe a number of approaches to user simulation. We distinguish between agenda-based and neural approaches, and between simulation at the semantic and at the word level.

2.1.2 Agenda-Based User Simulation

One common approach to user simulation is the Agenda-Based User Simulator (ABUS) proposed in Schatzmann et al. (2007). In this case, the simulator consists of a goal and an agenda. The goal is made up of a list of constraints and a list of requests. The constraints are slot-value pairs representing the user's preferences. An example of such a constraint in the restaurant booking domain would be `(food,Thai)`, indicating that the simulated user is looking for a restaurant serving Thai food. The requests are a list of slots that the user wants information on. For instance, the list of requests could be `(number, address)`, indicating that the simulated user needs to obtain the phone number and address of the selected venue.

The agenda is a list of dialogue acts needed to reach the user's goal, for example informing the system of the Thai food preference. The acts are removed from the agenda once they have been carried out by the simulator. The goal is constructed by randomly sampling a set of constraints and requests from a pre-defined domain ontology. The agenda is initially constructed from the goal, and at each turn one or more of the remaining acts are output by the system.

The ABUS can be entirely hand-crafted, but some of its parameters can also be estimated from data. Schatzmann and Young (2009) describe how the expectation maximisation algorithm can be used to achieve this. Keizer et al. (2010) propose a different approach to parameter estimation, introducing random decision points that can be encountered at each dialogue turn, and putting distributions on the available options at each point. The parameters of said distributions are then estimated from empirical counts. Unlike Schatzmann and Young (2009), this approach allows for changes in the user's goal during the dialogue.

Regardless of whether parameters are estimated from data, the agenda based approach has two major drawbacks. Firstly, it relies on a manually defined and domain specific rules, which can get intractable for complex tasks. Secondly, it simulates user behaviour at the semantic level (returning dialogue acts rather than natural language), which prevents the natural language understanding component of the dialogue system from being trained

[1] Clearly, the policy from the corpus cannot be completely dissociated from the one learnt by RL, since it influences the behaviour of the simulator, which itself will affect the types of situations that the system is exposed to during training.

jointly with the dialogue manager. Li et al. (2016) address this by adding a natural language generation module on top of an agenda based simulator. Another approach, which can potentially address both limitations, is to replace the agenda based generator with neural architectures. This possibility has been investigated by a number of recent papers, which we describe in the next section.

2.1.3 Neural User Simulation

Even when we restrict our attention to methods which rely on neural models, there remains a lot of variability in the approaches proposed in the literature. One obvious source of variability is model architecture, but models also vary in terms of their input and output formats. Some models are fully natural language to natural language, whereas others take in structured inputs from hand-crafted feature extractors, and others output only dialog acts, leaving the task of natural language generation to a separate module.

One of the first attempts at using neural methods for user simulation was made by Asri et al. (2016). Their simulator is based on an encoder and a decoder, both with long short term memory (LSTM) architectures. While their method arguably requires less manual intervention than agenda-based simulators, it is not fully natural language to natural language. Goal generation is done in the same way as in Schatzmann et al. (2007), and the decoder outputs sequences of dialogue acts rather than words. Furthermore, the input fed to the encoder is a sequence of context vectors encoding the state of the dialogue at each turn, with n-hot vectors indicating things like the latest system output. We have adopted their approach for feature extraction, so it is described in further detail in chapter 3.

Kreyssig et al. (2018) take a similar approach to Asri et al. (2016). They also use LSTMs for both the encoder and decoder, and adopt the same feature extractor. The main difference between the two approaches is that Kreyssig et al. (2018) produce natural language rather than semantics. As they point out, this means that the simulator can be trained on data which is not annotated with dialogue acts on the user side, as was the case for example for initial versions of the multiWOZ data set (Eric et al., 2019). Nie et al. (2019) take a similar approach, but replace the handcrafted feature extractor used in Kreyssig et al. (2018) with a graph convolutional network.

Before Kreyssig et al. (2018), Crook and Marin (2017) also produced a simulator generating natural language. Importantly, their model takes in natural language as input. The main drawback of their approach is that they do not define or condition on a specific user goal. This means that the simulator can't be used to train a policy, as the system would be able to dictate the user's goal and not be penalised for it. Consequently, Crook and Marin (2017)'s focus is on system evaluation rather than training. Gür et al. (2018) are also able to bypass

feature extraction, by using a hierarchical sequence-to-sequence model which first encodes the user goal and system actions, and then the full dialogue history. However, their proposed simulator outputs dialogue acts and not natural language. Hou et al. (2019) propose a state to sequence model which can output semantics or natural language, but needs structured inputs. They show that this requirement allows the model to perform better than a sequence to sequence alternative.

More recently, Lin et al. (2021) employ transformer architectures to construct a domain-independent simulator. The proposed model takes in structured input similar to Asri et al. (2016) and produces output in semantic form. Relative to existing neural simulators, the main improvement of the approach is therefore in the ease of domain adaptation. Tseng et al. (2021) also focus on domain adaptation. They propose a framework for joint optimisation of the dialogue system and user simulator, which relies on pre-training both through supervised learning, and then continued optimisation through interactions using RL methods. Other papers which jointly train the simulator and dialogue system include Takanobu et al. (2020), Liu and Lane (2017) and Papangelis et al. (2019).

We end this section by noting that, although neural methods have shown promise and generated a lot of excitement, as evidenced by the papers mentioned above, their superiority over agenda-based approaches is far from well established. For instance, Shi et al. (2019) compare them, combining them with various methods for natural language generation, and they find that agenda-based methods perform better, though they do not expand on why that might be the case.

2.1.4 Other Approaches

While agenda-based and neural approaches are the most relevant to our work, we briefly mention other methods that have been employed for user simulation. Eckert et al. (1997), Georgila et al. (2005), Pietquin and Dutoit (2006), and Eshky et al. (2012) all employ Markov models, conditioning on various amounts of context, with and without including user goal. Scheffler and Young (2001) model user intentions through a lattice structure, while Jung et al. (2009) employ a linear-chain conditional random field. Finally, Chandramohan et al. (2011) and Chandramohan et al. (2012) instead propose to look at user simulation as an inverse reinforcement learning problem, formulating user behaviour as a Markov decision process.

Despite the wide range of approaches present in the literature, to the best of our knowledge, no work has used GANs for user simulation in the context of task oriented dialogue systems. Before we introduce these models and related work in section 2.2, we must first discuss the important and rather difficult question of how to compare the performance of different simulators.

2.1.5 Evaluation of User Simulators

There is no obvious single method to compare the performance of two user simulators. Both Keizer et al. (2012) and Pietquin and Hastie (2013) provide surveys of the different metrics used in the literature.

We can distinguish two approaches to evaluation: direct and indirect evaluation. Direct evaluation involves computing statistical metrics to compare the simulator's output with data coming from human users. Indirect evaluation methods look at features of interactions between the simulator and dialogue systems. The most common of such methods is to use the simulator to train a dialogue system, and to use the performance of the trained system as the measure of the simulator's quality.

Among direct metrics, Pietquin and Hastie (2013) include some measures related to natural language generation, such as perplexity on test data, and bilingual evaluation understudy (BLEU). More relevant to simulation at the dialogue act level, which is the focus of this work, they also include measures of the divergence between the distributions of real and simulated user actions. These are precision, recall, balanced F-score, Kullback–Leibler (KL) divergence, and Discourse BLEU (D-BLEU). They are discussed in further detail when we describe our experimental methods in chapter 4.

Pietquin and Hastie (2013) also describe five indirect evaluation methods. The first is task completion, which lets the simulator interact with an existing dialogue system and measures how often the system correctly understands the goal, relative to what would happen if the simulator was performing random actions. The second measure looks at how well the simulator can predict the performance of different systems, by computing the Cramér-von Mises divergence between the distribution of performance scores of the system against the simulator and against real users. In the third strategy they propose to use statistical learning to develop metrics that correlate well with human evaluations, or, in other words, build models to predict human assessments. The fourth strategy, perhaps the most intuitive, is to evaluate dialogue policies trained with the simulator. As the authors point out however, this strategy requires testing a new dialogue system against real users, even though the goal of user simulation is precisely to avoid having to involve humans. We will come back to this difficulty in chapter 4. For the fifth and final evaluation strategy, they propose to train an optimal dialogue policy using the simulator, then take an existing dialogue corpus against human users, and, for each dialogue, measure how similar the policy used is to the learned optimal policy, and compute a measure of the quality of the dialogue (for instance using the rewards that would have been attributed by the RL algorithm). They then argue that if the simulator is realistic, there should be a positive correlation between the quality measure and the similarity between policies. Indeed, if the simulator behaves exactly like real users, the

optimal strategy against the simulator should also be optimal against real users. As they point out, the difficulty of this evaluation strategy lies in defining the similarity measure between policies.

Table 2.1 summarises the evaluation strategies adopted by the papers mentioned above[2]. It shows that a considerable amount of diversity exists in the literature, but that it is common for papers to combine several evaluation strategies. In particular, for papers focusing on building simulators for training (rather than, say, evaluation), one popular option is to combine a form of direct evaluation (e.g. F-score), with measuring the performance of a policy trained with the simulator. This is the approach we adopt, but we defer a detailed description to chapter 4. In the next section, we cover useful background related to GANs.

2.2 Generative Adversarial Networks for Sequences

We begin this section by explaining the idea behind GANs and the specificities of using them in the context of discrete and sequential data. We then discuss a number of papers that have used GANs for text generation, the application closest to ours.

2.2.1 Conditional Generative Adversarial Networks

GANs were first introduced by Goodfellow et al. (2014) for the task of image generation. We will work with conditional GANs, an extension proposed by Mirza and Osindero (2014). The idea behind GANs is to train two models in parallel: a generator and a discriminator. The generator is the model of interest, while the discriminator is simply used for training. Adversarial training can be thought of as a two player game between the generator and discriminator. The goal of the generator is to produce fake data which is similar to data drawn from a given population, e.g., a set of images, or a text corpus, and to fool the discriminator into classifying generated samples as real. The goal of the discriminator is to correctly classify samples as either generated or real. A high-level illustration of this architecture is presented in Figure 2.1.

Equation 2.1 gives the objective function corresponding to this adversarial training set up. θ_G and θ_D denote the parameters of the generator and discriminator, respectively. $p(x, y)$ is the real data distribution. x is information on which generation is conditioned. In our case, this will be dialogue contexts drawn from the data. y is the data we aim to imitate. z is the

[2]We exclude papers that train the simulator and dialogue system jointly from this table as their focus tends to be on measuring the quality of the dialogue system rather than the simulator.

Paper	Evaluation Strategies Used
Eckert et al. (1997)	None
Scheffler and Young (2001)	Task completion (# of turns)
Georgila et al. (2005)	Perplexity and task completion (% slots filled)
Pietquin and Dutoit (2006)	Performance of learnt policy (own-model evaluation)
Schatzmann et al. (2007)	Performance of learnt policy (human evaluation)
Schatzmann and Young (2009)	Performance of learnt policy (cross-model and human evaluation)
Jung et al. (2009)	BLEU, D-BLEU and KL-divergence
Keizer et al. (2010)	F-score, KL-divergence and performance of learnt policy (cross-model evaluation)
Chandramohan et al. (2011)	Task completion (reward and # of turns)
Chandramohan et al. (2012)	None
Eshky et al. (2012)	Perplexity
Li et al. (2016)	Performance of learnt policy (own-model evaluation)
Asri et al. (2016)	F-score
Crook and Marin (2017)	Perplexity and direct human evaluation
Kreyssig et al. (2018)	Performance of learnt policy (cross-model and human evaluation)
Gür et al. (2018)	Perplexity and task completion (% slots filled and # of turns)
Nie et al. (2019)	BLEU
Hou et al. (2019)	F-score and performance of learnt policy (cross-model and human evaluation)
Shi et al. (2019)	Perplexity, direct human evaluation and performance of learnt policy (human evaluation)
Lin et al. (2021)	F-score and performance of learnt policy (cross-model and human evaluation)

Table 2.1 Summary of evaluation strategies used in the literature

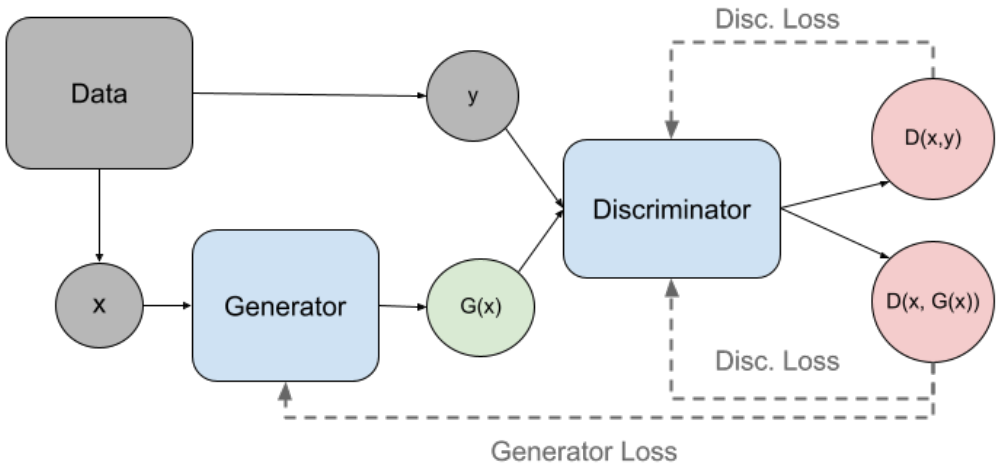


Fig. 2.1 Illustration of the conditional GAN architecture

synthetic data produced by the generator, and $p_G(z|x;\theta_G)$ is the generator output distribution. $D(\cdot)$ is the output from the discriminator.

$$\min_{\theta_G}\max_{\theta_D}\mathbb{E}_{x,y\sim p(x,y)}\log(D(x,y;\theta_D))+\mathbb{E}_{x\sim p(x),z\sim p_G(z|x;\theta_G)}\log(1-D(x,z;\theta_D)) \tag{2.1}$$

During training, we alternate between updating the parameters of the generator and of the discriminator according to equation 2.1. The updates to the generator aim to maximise the probability that the discriminator assigns to generated samples being actually drawn from real data. The updates to the discriminator correspond to the usual MLE training of a binary classifier.

2.2.2 Adversarial Training in the Discrete Domain

When generating images, or any other continuous output, both types of updates described in section 2.2.1 would be done using backpropagation. Unfortunately, this approach is unavailable when producing discrete outputs, as is the case in user simulation.

Indeed, the generator update involves propagating gradients through both the discriminator and the generator. When the generator output is discrete, for example from a sampling or max operation, this backpropagation is impossible. Several alternatives have been proposed, principally in the context of text generation. One possibility is to replace sampling from

the categorical distribution with a sample from a continuous approximation, e.g., a Gumbel-Softmax distribution (Jang et al., 2016; Maddison et al., 2016). Gulrajani et al. (2017) show that it is possible to train a character level language model by avoiding sampling altogether and feeding the softmax output directly to the discriminator. However, the most common approach, and the one that will be used to train our models, is to replace backpropagation with the REINFORCE algorithm (Williams, 1992), using the output from the discriminator as the reward. Equation 2.2 provides the corresponding gradient estimator.

$$\nabla_{\theta_G}\mathbb{E}_{x\sim p(x),z\sim p_G(z|x;\theta_G)}[D(x,z)] = \mathbb{E}_{x\sim p(x),z\sim p_G(z|x;\theta_G)}[D(x,z)\nabla_{\theta_G}\log(p_G(z|x;\theta_g))] \quad (2.2)$$

2.2.3 GANs for Text Generation

Although no work has used GANs for user simulation for training task oriented dialogue systems, a number of papers have focused on using them for text generation. While our application aims to generate dialogue acts rather than text, this is the most relevant part of the literature surrounding GANs. Indeed, our application shares a lot of the challenges and specificities of text generation. In both cases, the output is discrete, raising the issues discussed in section 2.2.2.

Another commonality is that both outputs are sequences, though generation at the word level tends to produce much longer sequences than at the dialogue act level. The sequential nature of the data raises the question of how to apply rewards to earlier parts of the sequence. Our description in section 2.2.2 treats the entire output as one whole, meaning that the same reward would be applied to every element of the output sequence. Several papers (de Masson d'Autume et al., 2019; Li et al., 2017) have found that they obtain better performance by providing the model with dense rewards, meaning rewards at every generation step. We will explore this in our experiments.

While several papers have explored using GANs for text generation, their advantage over models trained by MLE remains uncertain. For instance, Caccia et al. (2020) explore the trade-off between output quality (realism) and diversity for both types of models by adding and varying a temperature parameter to the softmax output layers. Using a variety of metrics, they argue that the MLE models outperform GANs in their applications.

One concern is that GANs may end up at a solution that is very close to the MLE one, while being generally more difficult and expensive to train. This problem is due to the fact that GANs are often pre-trained with MLE, because adversarial training can otherwise be very unstable. If a large amount of MLE pre-training is carried out relative to the adversarial fine-tuning, GANs will generally behave similarly to MLE models. de Masson d'Autume

et al. (2019) show that MLE pre-training can be avoided altogether, though they do not claim to improve on the performance of the MLE baseline, only to match it using only adversarial training. They make a number of useful recommendations for stabilising adversarial training, in particular providing the generator with dense rewards, using large batch sizes, and regularising the discriminator.

While de Masson d'Autume et al. (2019) provide useful guidance that we will refer to when training our own models, Li et al. (2017)'s goal is closer to ours. They use GANs for dialogue generation, though they do not focus specifically on task-oriented dialogues. Their system learns to respond to prompts drawn from movie subtitles with appropriate natural language. In terms of evaluation, they look both at using the success rate in fooling the discriminator, and at direct human evaluations of generated dialogues. They find that adversarial training performs better than MLE in terms of human evaluations, but they do not use a temperature parameter on the output, which means that their results are potentially vulnerable to the critique made by Caccia et al. (2020).

Finally, in recent years, several papers (Croce et al., 2020; Wu et al., 2021) have used adversarial fine-tuning on large pre-trained language models such as BERT (Devlin et al., 2019) and GPT-2 (Radford et al., 2019). As we have chosen to focus on user simulation at the dialogue act level, this avenue for research is less relevant and we do not discuss it further, but it could be an interesting approach to consider for future work focusing on natural language simulation.

Armed with a better understanding of both user simulation and GANs, we now describe how we combine them to create a GAN user simulator.

Chapter 3

The GAN User Simulator

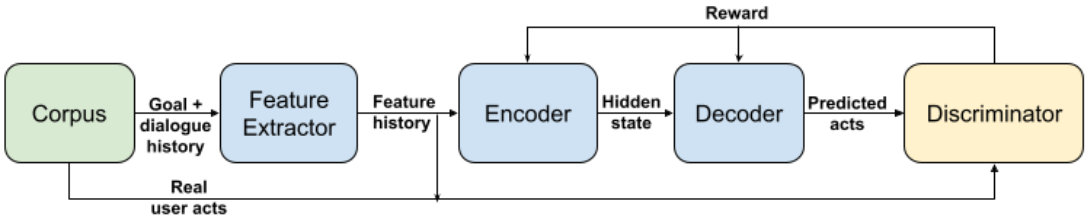
In this chapter, we describe the different components of our model. Most of them are also shared by the baseline MLE simulator as the models are identical except for the way they are trained. Therefore all of the descriptions below are also relevant to the baseline model, except for section 3.4 which discusses the discriminator.

Figure 3.1 shows how the different components of the simulator are related to each other during training and testing. It may be useful to refer back to it as we describe each component individually in the sections that follow.

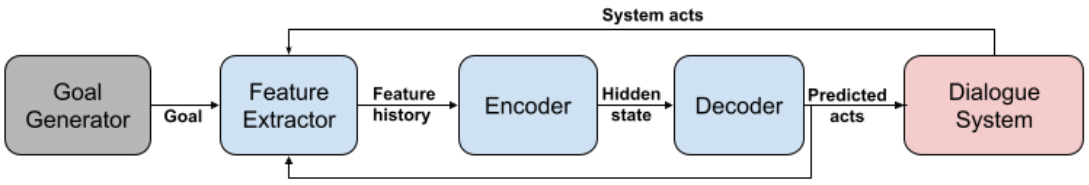
3.1 Goal Generator

During training, user goals are taken from the training examples in the corpus. We allow for goal changes in the same way as Kreyssig et al. (2018). The hidden goal associated with each turn contains all of the constraints and requests expressed throughout the dialogue, not just the ones that have been expressed so far. If the user expresses conflicting constraints, the goal is unchanged up until the conflict is expressed, and updated afterwards. So for instance if the user says at turn 2 that they would like a restaurant in the East, and at turn 4 that they would like one in the North, the goal associated with turns 0-3 inclusive will indicate the desired area as East, and 4 onward as North.

At test time, the user goal is randomly constructed from the domain ontology as follows. Initially, each informable slot, a slot which the user can express a preference over, has a probability of 0.4 of being excluded from the goal. If no slot is included, one is sampled at random. A value is picked at random from the ontology for each slot included in the goal. One (with probability 0.7) or two (with probability 0.3) of the requestable-only slots, slots which the user can want information about but not have preferences over, are included in the requests, chosen at random. Additionally, one of the informable slots excluded from the goal



(a) Training set-up



(b) Testing set-up

Fig. 3.1 Illustration of the overall training and testing set ups

is included in the requests with probability 0.3. The goal is resampled whenever the system says that the constraints cannot be met.

3.2 Feature Extractor

The simulator takes in structured inputs from the feature extractor representing the current dialogue context. This is not ideal, as it requires data to be labelled and makes domain adaptation more complicated, but it simplifies the simulator's task, relative to using natural language turns directly.

Our feature extractor is similar to the one used in Asri et al. (2016). Each dialogue turn has an associated context vector, which encodes the state of the dialogue before the user speaks as well as the underlying user goal. The context vector is made up of four subvectors, whose elements are all binary variables.

First, the constraints vector $const_t$ encodes which of the informable slots still need to be communicated by the user. Any constraint which is not part of the user goal is set as informed from the start, which is denoted with value 1. Constraints that are part of the goal are initially set as to be informed and encoded as zeros, and switched to ones as the user communicates them to the system. When the system requests information about a slot, or misunderstands a constraint expressed by the user, the corresponding element in the constraints vector is changed to 0.

Similarly, the requests vector req_t encodes which slots are still to be requested. Slots are tagged as filled as the system informs the user of their value for a specific venue under consideration. When discussion moves on to a new venue, the requests vector is reset to its original value. Unfortunately this means we assume that there is only ever one venue under discussion at a time, and rules out things like referring expressions.

The third vector is the inconsistency vector, inc_t. This vector is usually all zeros, except when one of the system actions from the latest turn indicates that the system has misunderstood one of the user constraints. In that case the corresponding element is changed to 1.

The final component of the context vector, act_t encodes the latest system action(s). There are 16 possible system actions, listed in appendix Table A.1. They are a combination of action type and, in some cases, slot.

The overall context vector, x_t is then the concatenation of all four vectors:

$$x_t = const_t \oplus req_t \oplus inc_t \oplus act_t \tag{3.1}$$

where $\oplus$ denotes concatenation.

We now illustrate the feature extractor by showing an example dialogue and corresponding context vectors.

Example Dialogue

```
USER GOAL
  Constraints:  pricerange=moderate, area=south
  Requests:  address

DIALOGUE
  Turn 0
    SYSTEM - Welcome to the Cambridge restaurant system.  How may
    I help you?
    USER - I want a restaurant in the South.

  Turn 1
    SYSTEM - What kind of food would you like?
```

$$x_0 = \begin{bmatrix} 1. \\ 0. \\ 0. \\ 1. \\ 1. \\ 1. \\ 1. \\ 0. \\ 1. \\ 1. \\ 1. \\ 0. \\ 0. \\ 0. \\ 1. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \end{bmatrix} = \begin{bmatrix} \text{inform food} \\ \textbf{inform area} \\ \textbf{inform pricerange} \\ \text{request food} \\ \text{request area} \\ \text{request pricerange} \\ \text{request name} \\ \textbf{request address} \\ \text{request phone} \\ \text{request postcode} \\ \text{request signature} \\ \text{inconsistency food} \\ \text{inconsistency area} \\ \text{inconsistency pricerange} \\ \textbf{act welcomemsg} \\ \text{act offer} \\ \text{act inform} \\ \text{act canthelp} \\ \text{act canthelp exception} \\ \text{act confirm domain} \\ \text{act reqmore} \\ \text{act impl-conf} \\ \text{act expl-conf} \\ \text{act repeat} \\ \text{act request food} \\ \text{act request area} \\ \text{act request pricerange} \\ \text{act select food} \\ \text{act select area} \\ \text{act select pricerange} \end{bmatrix} \tag{3.2}$$

$$x_1 = \begin{bmatrix} 0. \\ 1. \\ 0. \\ 1. \\ 1. \\ 1. \\ 1. \\ 0. \\ 1. \\ 1. \\ 1. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 1. \\ 0. \\ 0. \\ 0. \\ 0. \\ 0. \end{bmatrix} = \begin{bmatrix} \textbf{inform food} \\ \text{inform area} \\ \textbf{inform pricerange} \\ \text{request food} \\ \text{request area} \\ \text{request pricerange} \\ \text{request name} \\ \textbf{request address} \\ \text{request phone} \\ \text{request postcode} \\ \text{request signature} \\ \text{inconsistency food} \\ \text{inconsistency area} \\ \text{inconsistency pricerange} \\ \text{act welcomemsg} \\ \text{act offer} \\ \text{act inform} \\ \text{act canthelp} \\ \text{act canthelp exception} \\ \text{act confirm domain} \\ \text{act reqmore} \\ \text{act impl-conf} \\ \text{act expl-conf} \\ \text{act repeat} \\ \textbf{act request food} \\ \text{act request area} \\ \text{act request pricerange} \\ \text{act select food} \\ \text{act select area} \\ \text{act select pricerange} \end{bmatrix} \tag{3.3}$$

Equation 3.2 shows the context vector in the first turn, after the system greets the user, but before the user says anything. Indicators corresponding to the two constraints present in the user goal, area and price range, are set to 0, as is the one relating to the single request to be made, the address. There are no misunderstandings, so the inconsistency vector is entirely zeros. In the act vector, only the element corresponding to the welcome act is set to 1.

Equation 3.3 shows the second context vector. In the constraints vector, the area has been informed so is set to 1, whereas the element corresponding to the food slot is set to 0, because it has been requested by the system (even though it is not present in the user goal, meaning that the user is indifferent towards it). The only other change is in the acts vector, where the element corresponding to the act `request food` is now the only non-zero element.

3.3 Generator

The generator takes as input the sequence of past context vectors, and outputs a sequence of dialogue acts. Figure 3.2 illustrates its overall architecture. It is made up of an encoder and a decoder. Both are Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNNs), with a single LSTM layer.

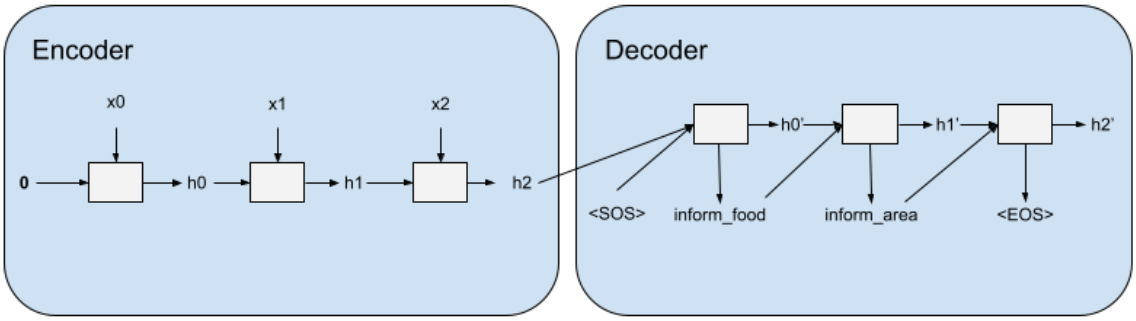


Fig. 3.2 Illustration of the generator architecture

3.3.1 LSTMs

The LSTM layer is recurrent, meaning that, while processing a sequence, at every time step, the network takes in as input the network output from the previous time step. The particularity of an LSTM unit is that it possesses a cell, c_t, which can record information from previous parts of the input sequence. Three gates regulate the flow of information through the network. The forget gate, f_t, regulates how much of the previous cell value is retained. The input gate, i_t regulates the extent to which new information is written into the cell. The output gate o_t regulates the link between the current cell and the hidden state h_t. Equations 3.4-3.9 formalise the relationship between each of these elements, where σ denotes the sigmoid function, and $\odot$ the Hadamard product. Ws are weight matrices to be trained.

$$i_t = \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right) \tag{3.4}$$

$$f_t = \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right) \tag{3.5}$$

$$g_t = \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right) \tag{3.6}$$

$$o_t = \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right) \tag{3.7}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{3.8}$$

$$h_t = o_t \odot \tanh\left(c_t\right) \tag{3.9}$$

3.3.2 Encoder

The input sequence to the encoder is the sequence of all past context vectors, up to the dialogue turn for which we want to predict the user's actions. The hidden state has size 32, and is initialised to the zero vector. The final hidden and cell states are kept and fed to the decoder.

3.3.3 Decoder

The first decoder layer is an LSTM layer with hidden state of size 32, just like the encoder. It is followed by a linear layer:

$$y_t = h_t W + b \tag{3.10}$$

$$\tag{3.11}$$

and a softmax activation:

$$p_{it} = \frac{\exp(y_{it})}{\sum_j \exp(y_{jt})} \tag{3.12}$$

where i denotes one of the possible user acts.

The hidden and cell states of the decoder are initialised to the final values from the encoder. It is given a start of sequence token as input. This is fed through the LSTM layer, the linear layer and the softmax activation. The latter gives a probability distribution over 28 dialogue acts (listed in Table A.2), plus an end of sequence token. One of these is sampled from this distribution. Then one of two things can happen. Either the dialogue act from real user data is fed as the next element in the input sequence to the decoder. This is called teacher forcing. Or, the sampled act is used instead. This process is repeated until the end of sequence token is sampled (or reached, if using teacher forcing).

3.4 Discriminator

The discriminator is a simple 2-layer feed forward network with a softmax on the output. The first layer has 64 units, and the output layer has two. The input to the discriminator is the concatenation of a context vector and following user response. The user response can either come from the data, or have been predicted by the generator, and it is the task of the discriminator to distinguish between these.

To transform the sequence of user acts into a single vector, one hot encodings of each user act in the sequence are concatenated, and zero padding is used to keep the length fixed. We have experimented with using an RNN for the discriminator, to allow it to take into account the full dialogue history, but we have found that doing so does not affect performance.

3.5 Training

We carry out two types of training, adversarial, and MLE. The latter is used mainly for training a baseline model, but will also be used as pre-training for GANs. In both cases, we use the Adam optimiser (Kingma and Ba, 2015) with default PyTorch settings, and teacher forcing with probability 0.5.

Adversarial training proceeds as follows. A dialogue turn is sampled from the data. The dialogue history and hidden goal are fed to the feature extractor. The resulting sequence of context vectors is fed to the generator, which outputs a sequence of predicted dialogue acts. The real and predicted dialogue acts are separately fed to the discriminator, along with the last context vector. For each, the discriminator outputs a probability that the example came from real data, and its parameters are updated via maximum likelihood using the Adam optimiser. The probability assigned to the predicted example is used as a reward for the generator, which is updated using equation 2.2 (REINFORCE). In chapter 5, we will investigate how various aspects of this process can be modified, and describe their effect on model performance. First, we describe our experimental methods in detail.

Chapter 4

Experimental Methods

This section describes the data and metrics used in our experiments.

4.1 The DSTC-2 Data Set

We base our experiments on the corpus provided for the second Dialogue State Tracking Challenge (DSTC-2) (Henderson et al., 2014). The training and development sets consist of a total of 2,118 dialogues between human users and a dialogue system. The test set contains 1,117 dialogues, which we reserve for evaluation in section 6.1. Each user is looking for a restaurant which satisfies a number of constraints, and wants to make a number of information requests relative to the chosen establishment.

Eight pieces of information can be requested by users: area, food, price range, name, address, phone number, postcode, and signature dish. Of these, only area, food and price range can be specified as constraints in the user's preferences.

The data is provided with semantic labels, meaning that each natural language utterance made by either the user or the system is associated with a dialogue act, consisting of an act type, and zero or more associated slots and values.[1] For instance, if the user said "I would like to find an Italian restaurant", the associated act type would be `inform`, the slot `food` and the value `Italian`.

These dialogue acts are used to extract features input to the simulator, and as targets for the simulator to predict (in the case of dialogue acts performed by the user). Throughout our experiments, we simplify the dialogue acts by omitting the values, and filling them in as needed from the user goal or dialogue context. This means that in the example given above, the simulator's task would have been to predict an act `inform` and slot `food`, but

[1] On the user side, these are manually annotated. On the system side, they are taken from system logs.

the associated value `Italian` would have been filled in using the user goal. This approach has the advantage that values in the domain ontology can be modified without changing the simulator.

4.2 Direct Evaluation Metrics

We carry out two types of model evaluation, direct and indirect. In direct evaluation, we measure how closely the simulated behaviour replicates real user behaviour when provided with contexts sampled from DSTC-2.

4.2.1 F-score

The F-score (F) measures how well the simulator predicts user dialogue acts from training examples. It combines precision (P) and recall (R) as defined in equations 4.1, 4.2 and 4.3.

$$P = \frac{\text{number of correctly predicted dialogue acts}}{\text{number of predicted dialogue acts}} \tag{4.1}$$

$$R = \frac{\text{number of correctly predicted dialogue acts}}{\text{number of dialogue acts in real user data}} \tag{4.2}$$

$$F = \frac{2PR}{P+R} \tag{4.3}$$

4.2.2 KL-Divergence

The KL-divergence (Kullback and Leibler, 1951) measures how similar one probability distribution P is to another, reference distribution Q. For discrete probability distributions, it is defined as:

$$D_{KL}(P||Q) = \sum_{i=1}^{M} p_i \log(\frac{p_i}{q_i}) \tag{4.4}$$

In the context of dialogue simulation, we take p_i to be the relative frequency of dialogue act i in user data, and q_i its relative frequency in simulated data.

While we will sometimes consider overall dialogue act distributions, we will also be interested in what we call context specific or conditional KL-divergence. In this case, we will subset the data to only include user actions and simulator predictions in response to a

specific context vector (defined in section 3.2), and compute the KL-divergence based on these frequencies. We then report the average of the resulting divergences over all contexts.[2]

4.3 Indirect Evaluation

The direct metrics described above may be useful indicators of simulator quality, especially because they can be computed quickly and can therefore be used during simulator development. However, the end goal of user simulation is to train good dialogue systems. For that reason, the best measure of simulator quality will be the performance of a policy trained against it. Accordingly, we train policies using both the GAN and the MLE simulator, and compare their performances.

4.3.1 Policy Training

We interact Toshiba's existing dialogue system with both simulators. Each simulator is used to train 10 different policies with different initialisations. Training is done via reinforcement learning (Monte-Carlo control with linear value function approximation (Sutton and Barto, 2018)) with learning rate of 0.01, which is linearly decayed by a factor of 2.475E-07 after every dialogue. We train each policy for 40,000 dialogues. Dialogues are limited to 30 turns. The agent receives a reward of +100 upon successful task completion, and -1 for each dialogue turn. There is also a "social penalty" of -5 applied when the system is thanked or said farewell to, and does not respond appropriately. During training and testing, we include an error model so that the system misunderstands the user dialogue act with probability 0.25. A dialogue is considered successful if the user says goodbye while all requests made by the user have been fulfilled by the system, and the restaurant currently under discussion fulfils all the user's expressed constraints. All interactions are done in dialogue acts, and no natural language generation or understanding modules are used.

4.3.2 Policy Evaluation

Ideally, policy performance should be measured against human users. While this may seem to defeat the purpose of user simulation by reintroducing humans into system development, evaluation requires far fewer interactions than training, making this approach sensible. Indeed, it is adopted by many papers as shown in Table 2.1.

[2] In practice, we cannot compute this for all contexts, as some appear too rarely. We restrict our attention to contexts which appear at least 20 times in whatever subset of the data is being used to compute the divergences.

Unfortunately, we have not had enough time to involve human participants in the evaluation of our dialogue systems. Instead, we rely on cross-model evaluation, meaning that we evaluate the learnt strategies against simulators other than the one they are trained on. Schatzmann et al. (2005) have shown that models trained on "good" simulators will tend to perform well even when tested against a variety of simulators. On the other hand, policies trained against poor simulators may perform well against their own simulator, because of overfitting, but not against others.

We use three simulators for evaluation: the GAN and MLE simulators used in training, and an agenda-based simulator, which is used exclusively for evaluation. We report success rates, average rewards and average dialogue lengths averaged over 3,000 evaluation dialogues.

Chapter 5

Adversarial Training Experiments

In this section, we describe how various choices regarding adversarial training affect the performance of the simulator. For these experiments, we use only direct evaluation metrics. We then select one GAN simulator, based on its F-score on validation data, to compare against the MLE simulator by direct and indirect methods in chapter 6.

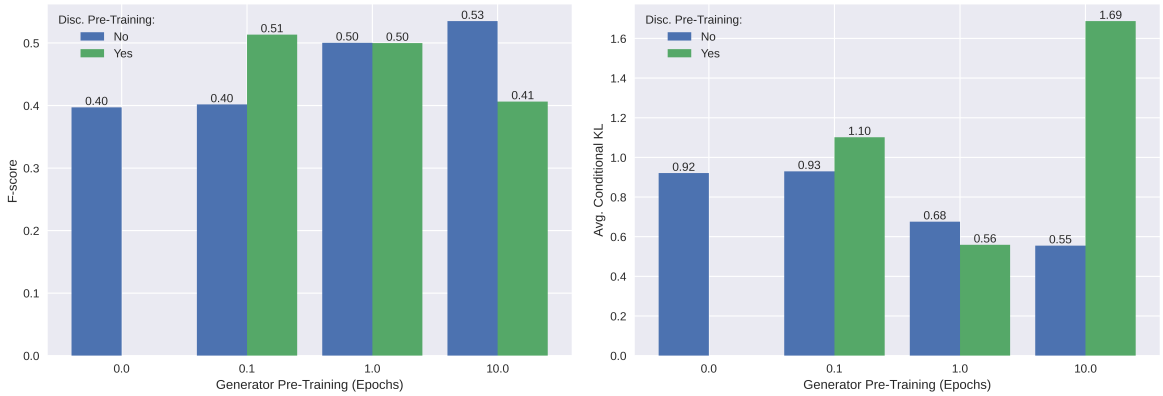
5.1 MLE Pre-Training

As mentioned in chapter 2, it is common to pre-train GANs for text generation using maximum likelihood (Caccia et al., 2020; Li et al., 2017; Yu et al., 2017). We investigate how different amounts of pre-training affect the performance of the GAN simulator model.

We pre-train the generator, and optionally the discriminator, for various numbers of epochs via MLE, then further train each model via adversarial training for 20 epochs. We then report the maximum F-scores and context specific KL-divergences over the adversarial training epochs.

The results are shown in Figure 5.1. As expected, MLE pre-training of the generator improves performance relative to using adversarial training exclusively. Most of the benefits of pre-training are gained after a single epoch, though we do observe further improvements from additional pre-training, especially in terms of KL-divergence.

The effect of pre-training the discriminator is ambiguous. It has a large positive effect on the performance of the model with 0.1 epochs of pre-training, but a large negative effect on the performance of the model with 10 epochs of pre-training. One possible explanation is that the fully trained discriminator has had enough time to partially memorise the training data. In that case the rewards it provides will be very close to one for replicating training examples exactly, and zero otherwise. This lack of smoothness in the reward signal may make learning difficult for the generator.



(a) F-scores (b) Conditional KL-Divergences

Fig. 5.1 Effect of MLE pre-training on F-score (left) and average conditional KL-divergence (right). Both measures are explained in chapter 4 and are calculated over a validation data-set. Each model is pre-trained by MLE for the number of epochs given by the x-axis, and then trained using adversarial training for a further 20 epochs, using Adam, a learning rate of 0.001 and batch size 1. The metric from the best epoch is shown. When the discriminator is also pre-trained (green bars) it is pre-trained for the same number of iterations as the generator.

Figure 5.2 shows the training history for the models with 0.1 and 1 epochs of pre-training. We can see that the adversarial training process is very unstable, with performance being far from monotonically increasing. The experiments in the remainder of this section attempt to stabilise training.

Though the models with 0.1 and 1 epochs of pre-training perform similarly in terms of maximum F-score, Figure 5.2 shows that they reach that point in very different ways. After a full epoch of MLE pre-training, the simulator has already reached close to its highest F-score, and adversarial training does not improve it significantly. For the model with only 0.1 epochs of pre-training on the other hand, adversarial training determines most of the performance. To ensure that the comparison we make in the next chapter between the GAN and MLE simulators is meaningful, and does not compare two mainly MLE trained simulators, the remainder of this chapter focuses on models with 0.1 epochs or no MLE pre-training.

5.2 Larger Batch Sizes

The next three sections experiment with recommendations made in de Masson d'Autume et al. (2019). First, we look at increasing batch sizes to stabilise training. Table 5.1, shows the model performance when we vary batch size during adversarial training, for the GANs

Fig. 5.2 Adversarial training history for models with different levels of pre-training. This graph shows F-scores (left scale) on validation data after each epoch of adversarial training, for the models with 0.1 and 1 epochs of MLE pre-training. We also include adversarial loss (right scale), averaged over 1,000 training examples at a time. Other training details are as described for Figure 5.1.

	No Pre-Training		0.1 Epoch Pre-Training	
Batch Size	F-score	Cond. KL	F-score	Cond. KL
1	0.40	0.92	0.51	1.10
5	0.42	0.77	0.42	0.75
10	0.41	0.77	0.40	0.65
15	0.42	0.58	0.41	0.68
20	0.38	0.61	0.40	0.72

Table 5.1 Effect of batch size on F-score and average conditional KL-divergence. The model with pre-training has pre-training both for the generator and discriminator. The best statistic over 30 epochs of adversarial training calculated over the validation data set is shown.

with no pre-training and 0.1 epochs of pre-training. As explained above, we do not want to rely on large amounts of MLE pre-training, which is why we focus on these two models.

Overall, Table 5.1 shows that using batch sizes larger than 1 is helpful, especially in terms of conditional KL-divergence, but the effect is not large, and is ambiguous for the F-score. Figure 5.3 shows that, even though the effect on final model performance is not large, increasing the batch size does stabilise training, as desired. It shows training history for the model with no MLE pre-training, trained with batch size of 1 and 15. The F-scores reported for the model trained with batch size 1 are clearly more noisy than those reported for the model trained with batch size 15.

5.3 Dense Rewards

As mentioned in chapter 2, a practice that some papers (de Masson d'Autume et al., 2019; Li et al., 2017) find useful when training GANs for text generation is to provide the model with rewards for partially decoded sequences, rather than assigning the same value to each word.

We have implemented this by training the discriminator to distinguish real and synthetic data based on sub-sequences of dialogue acts. During adversarial training, for each training example, we randomly truncate both the real and generated sequence by 0 or more dialogue acts. The discriminator outputs a reward based on the chosen sub-sequence, which is then used to update both generator and discriminator parameters in the usual way.

Figure 5.4 shows how this affects performance for the model with no MLE pre-training and trained using a batch size of 15. Clearly, providing the generator with dense rewards improves training, especially in early epochs. The model trained with dense rewards achieves F-scores comparable to the models with 1 or 10 epochs of MLE pre-training shown in Figure

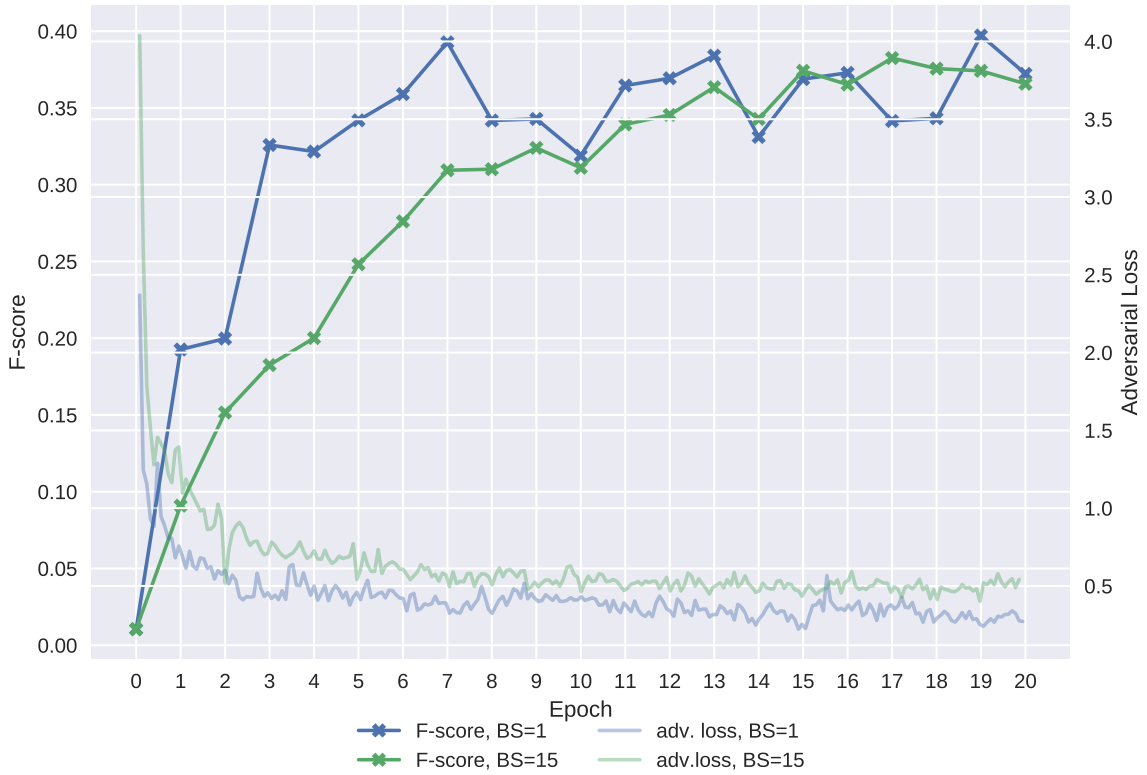


Fig. 5.3 Adversarial training history for different batch sizes. This graph shows F-scores (left scale) on validation data after each epoch of adversarial training, when using batch size of 1 and 15, for the no MLE pre-training. The right scale shows adversarial loss averaged over 1,000 training examples at a time.

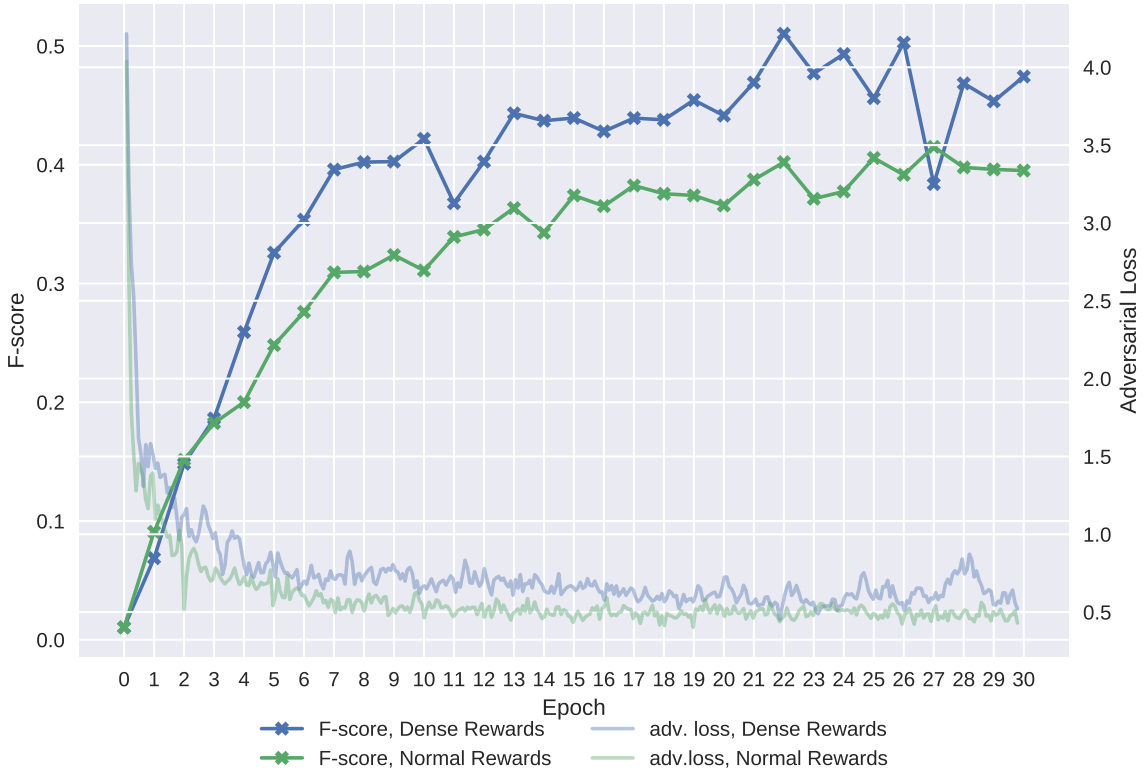


Fig. 5.4 Adversarial training history for models with and without dense rewards. This graph shows F-scores (left scale) on validation data after each epoch of adversarial training. We also include adversarial loss (right scale), averaged over 1,000 training examples at a time. Both models use batch size of 15, learning rate of 0.001, and no MLE pre-training.

5.1, despite not relying on any pre-training itself. We therefore keep using dense rewards and remove pre-training in further experiments.

5.4 Discriminator Regularisation

de Masson d'Autume et al. (2019) recommend adding regularisation to the discriminator. We therefore experimented with various amounts of L_2 regularisation. This type of regularisation adds the L_2 norm of the weight matrix to the discriminator's loss function, multiplied by some weight γ, thus encouraging simpler models with weights closer to 0. Larger values for γ put a larger penalty on complex models.

As shown in Figure 5.5, we have experimented with different levels for γ but none improved performance. The discriminator is potentially already simple enough, with only 2 hidden layers and 7,682 parameters in total.

#gen. updates/disc. update	F-score	Conditional KL-divergence
0.5	0.51	1.21
1	0.50	1.28
2	0.50	1.17

Table 5.2 Effect of update schedule on performance. None of the models have any MLE pre-training. For each model we use batch size 15, dense rewards, learning rate of 0.001. The best statistics over 30 epochs of adversarial training are shown.

5.5 Update Schedule

The final decision relating to adversarial training which we discuss is the relative frequency with which we update parameters from the generator versus the discriminator. In our default training set up, the discriminator's parameters are updated twice as often as the generator, because for every training example, we provide the discriminator separately with the true and the synthetic user acts, and update parameters based on the loss for each. This may not be a reasonable choice, especially given that the discriminator has only 7,682 parameters, compared with the generator's 13,695.

We therefore tried increasing the relative number of updates to the generator parameters, but did not find an improvement in performance. The results can be seen in Table 5.2

5.6 Discussion and Model Selection

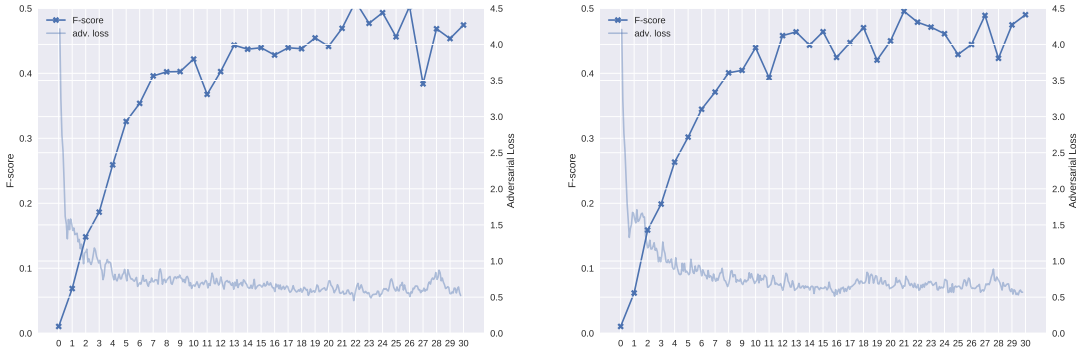
Overall, our experiments corroborate findings present in the literature for text-generating GANs. We have found that pre-training the generator by MLE improved its performance. We have also found that the recommendations made by de Masson d'Autume et al. (2019) for stabilising adversarial training, specifically using large batch sizes and dense rewards, were helpful in our application. One exception to this is adding regularisation to the discriminator, which did not improve our results.

These improvements to the adversarial training process allow us to avoid MLE pre-training altogether, while still obtaining a model which performs well in terms of F-score. We think avoiding MLE pre-training is valuable because it makes it more likely that we will obtain a simulator which is truly different from the baseline MLE trained one.

In the next chapter, we therefore focus on the best performing GAN that has no MLE pre-training, where performance is measured by F-score on validation data. Table 5.3 summarises the chosen model's training settings.

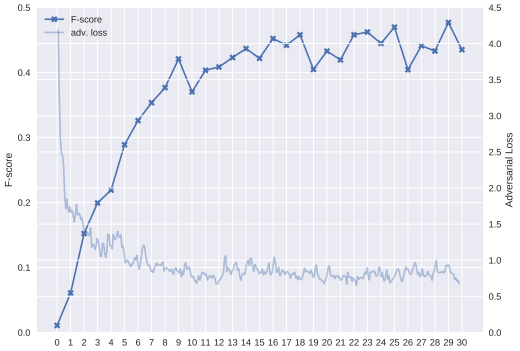
Parameter	Value
MLE pre-training epochs	0
Batch size	15
Dense rewards	True
#gen. updates/disc. update	0.5
Weight decay on discriminator	0
Learning rate	0.001
# of adv. training epochs	30

Table 5.3 Summary of the selected GAN simulator's training parameters.

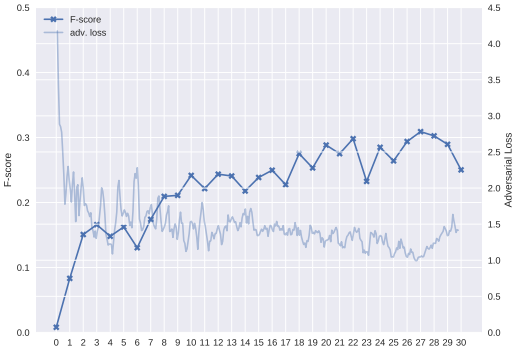


(a) $\gamma = 0$

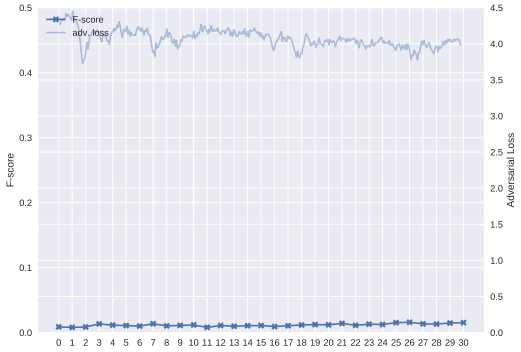
(b) $\gamma = 0.0001$



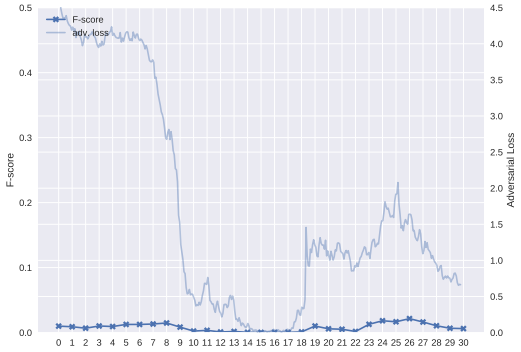
(c) $\gamma = 0.001$



(d) $\gamma = 0.01$



(e) $\gamma = 0.1$



(f) $\gamma = 1$

Fig. 5.5 Training history for varying weight decay parameters. Each figure shows F-score on validation data (left scale) and adversarial loss (right scale). The models use dense rewards, batch size 15, learning rate of 0.001 and no MLE pre-training.

Chapter 6

Comparison with an MLE Simulator

In this chapter we compare the performance of the GAN simulator selected through the experiments of chapter 5 with that of a simulator trained with MLE. We first evaluate both simulators on the direct metrics used in the previous chapter, calculated on a held-out test data-set. We then use both simulators to train dialogue policies, and report on the performance of these using cross-model evaluation.

6.1 Direct Evaluation

We begin by comparing the two simulators using the direct evaluation methods described in chapter 4, measured on a held out test set[1]. Figure 6.1 shows the two simulators' performances in terms of F-score, average conditional KL-divergence and overall KL-divergence. On each of these metrics, the GAN simulator is outperformed by MLE.

While the results shown in Figure 6.1 are not encouraging, all three metrics included are related to the maximum likelihood criterion, for which the MLE simulator is optimised. Therefore the superiority of the MLE simulator on these measures is not surprising, and may not be that meaningful. As previously discussed, the F-score and KL-divergence may be helpful indicators of simulator quality, but they do not directly measure performance on the end goal of user simulation, which is to train high performance dialogue systems. This is why we now turn to indirect evaluation methods to compare the two simulators.

[1] Our test set lines up with the official DSTC-2 test set.

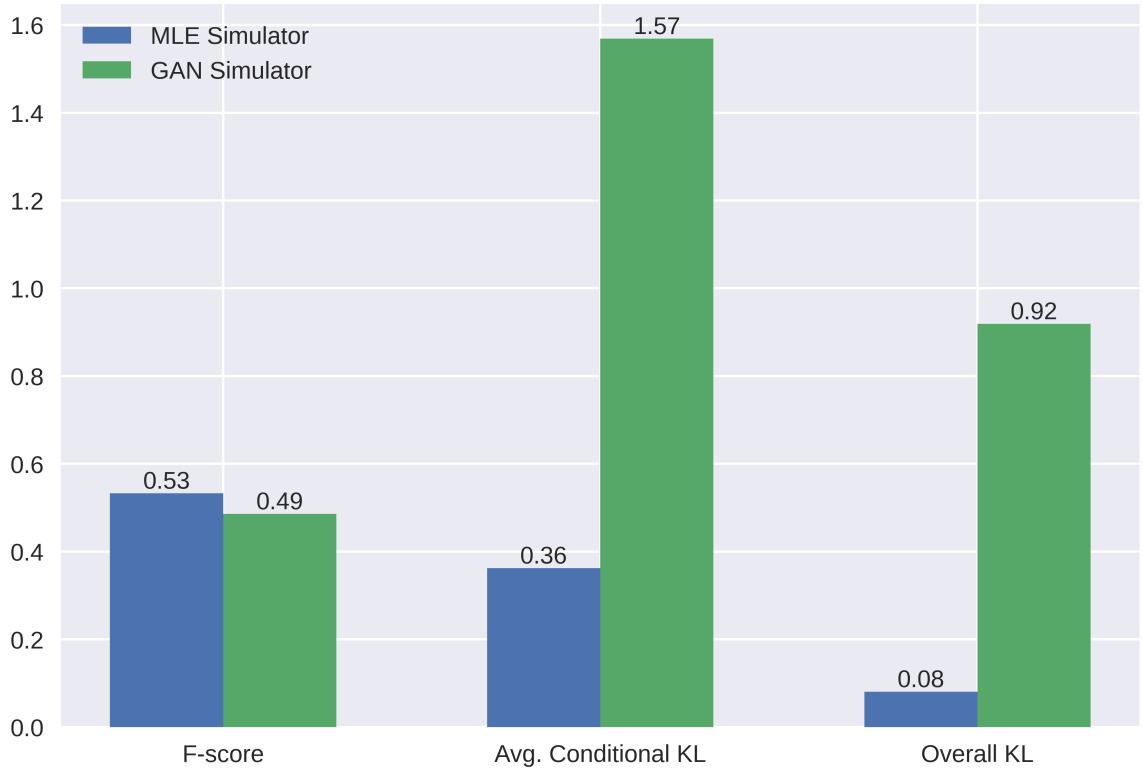


Fig. 6.1 Direct comparison of the GAN and MLE simulators. The GAN simulator's training parameters are as shown in Table 5.3. The MLE simulator is trained for 20 epochs. The metrics are as defined in chapter 4, and we measure them on a held out test set (separate from the validation set used in chapter 5).

6.2 Indirect Evaluation

We train 10 dialogue policies via reinforcement learning, against each of the simulators. Training details are as described in chapter 4. We train several policies per simulator to account for the randomness in different training runs. Average success rate of the resulting

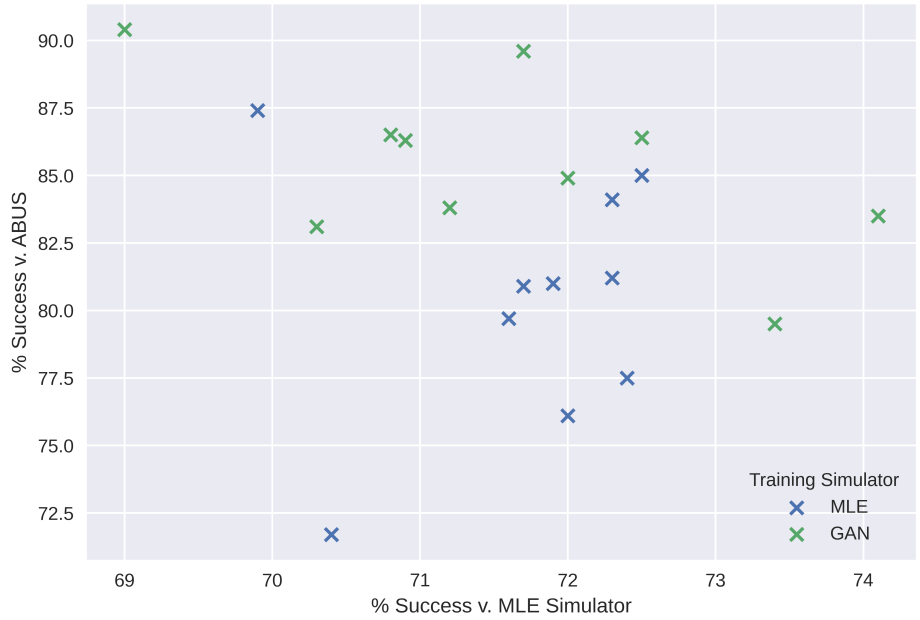
		Evaluation Simulator		
		MLE	GAN	ABUS
Training	MLE	**71.70**	70.05	80.46
Simulator	GAN	71.59	**71.52**	**85.40**

Table 6.1 Average success rate against different simulators for policies trained with the MLE and GAN simulators. Success rates are calculated over 3,000 evaluation dialogues, and averaged over 10 training runs for each training simulator. Policies are trained for 40,000 dialogues.

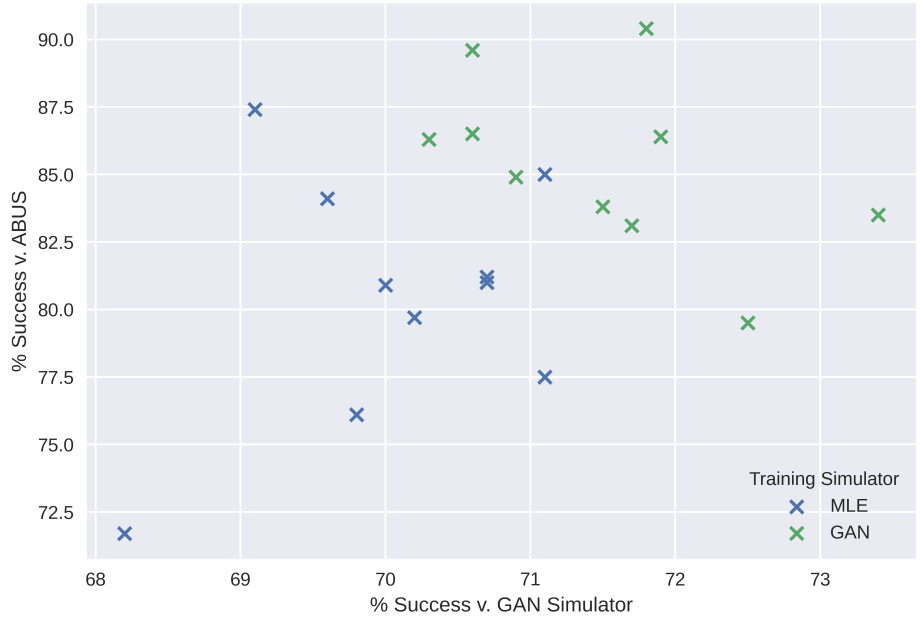
policies against all three simulators is shown in Table 6.1. Average rewards and number of turns per dialogue are included as appendix tables. As could have been expected, on average the policies that do best against the MLE simulator are the ones that were trained against it, and likewise for the GAN simulator, though the differences are small, especially against the MLE simulator. Against the ABUS, which should not particularly favour either training method, policies trained against the GAN simulator perform better on average.

In Table 6.2 we report the success rate of the best performing policies for each training-evaluation simulator pair. Against all three evaluation simulators, the best performing policy is one trained against the GAN simulator.

There exists some variance in performance among policies trained against the same simulator. This can be seen in Figure 6.2 where we show policy performance against the ABUS versus against the MLE simulator in (a) and against the ABUS versus against the GAN simulator in (b). We also include these results as appendix tables. Figure 6.2 shows that there is a negative correlation between performance against the GAN simulator and against the ABUS, for policies trained against the GAN simulator. This raises the concern that the policies are overfitted to their training simulator. We do not believe that this is actually the case, firstly because there is a positive correlation between their performance against the GAN and against the MLE simulator, and secondly because we have looked at the performance against the ABUS of policies trained for only 20,000 dialogues and found that it was significantly worse.



(a) Success rates against ABUS and MLE simulators



(b) Success Rate against ABUS and GAN simulators

Fig. 6.2 Success rate of individual policies, measured over 3,000 evaluation dialogues. Policies are trained for 40,000 dialogues.

A potential worry when looking at these results is that the two simulators may not be doing anything fundamentally different from each other, but that the GAN simulator has been trained slightly less efficiently, and thus fits the data less closely, and that the extra randomness in its output is helpful in training. If that is indeed the case, we should be

		Evaluation Simulator		
		MLE	GAN	ABUS
Training	MLE	72.5	71.1	87.4
Simulator	GAN	**74.1**	**73.4**	**90.4**

Table 6.2 Maximum success rate over 10 policies trained by each simulator. Success rates are calculated over 3,000 dialogues. Policies are trained for 40,000 dialogues.

		Evaluation Simulator		
		MLE	GAN	ABUS
Training	MLE	**71.70**	70.05	80.46
Simulator	w/ early stopping	71.16	**72.07**	**82.37**

Table 6.3 Average success rate against different simulators for policies trained with the fully trained MLE simulator, or the MLE simulator trained for a single epoch (early stopping). Success rates are calculated over 3,000 evaluation dialogues, and averaged over 10 training runs for each training simulator. For evaluation, the fully trained MLE simulator is used.

concerned that the same level of performance could be achieved by an MLE simulator with heavy regularisation, or early stopping, or a temperature parameter on the softmax output.

Investigating this issue fully would have been costly, as it would ideally involve a temperature sweep, and training dialogue policies is time-consuming. We are therefore not in a position to completely rule out this possibility, but we did train dialogue policies using an MLE simulator with only one full epoch of training. This model already fits the data reasonably closely, with an F-score of 0.48 and conditional KL-divergence of 0.33. The dialogue policy training details are as before, meaning that early stopping is only used for simulator training, not policy training. We look at performance of the trained policies against all three simulators in Table 6.3. Performance against ABUS is still worse than that observed for policies trained against the GAN simulator, but it is improved slightly relative to using the fully trained MLE simulator. Interestingly, even when evaluating against the fully trained MLE simulator, we do not observe much deterioration in average performance from training with the partially trained MLE simulator.

Overall then, stopping the MLE simulator training early results in slightly better policies, even though the simulator performs worse in terms of direct evaluation metrics. This once again highlights the fact that metrics such as the F-score and KL-divergence may not be good indicators of simulator quality for the purposes of policy training.

Generally, the results presented in this section are encouraging. They suggest that the GAN simulator may be a better choice to train dialogue systems than MLE simulators. There are two main caveats to this conclusion. Firstly, as we have just discussed, we cannot entirely rule out that a regularised MLE simulator might perform just as well. Secondly, we will need to confirm these results by carrying out policy evaluations against human users.

Chapter 7

Conclusion

7.1 Summary

In this dissertation, we have built and evaluated a user simulator for task oriented dialogue systems based on a generative adversarial network.

Our proposed simulator generates dialogue acts, and does not produce any natural language. It relies on a handcrafted feature extractor which produces context vectors that summarise the state of the dialogue at each turn. The main component of our simulator is the generator, which is made up of an LSTM encoder and an LSTM decoder. Based on a sequence of context vectors corresponding to a specific dialogue turn, it predicts a sequence of dialogue acts that form the user's response. The generator is trained adversarially, through interactions with a discriminator. Its objective is to produce output that the discriminator is unable to distinguish from human user data.

We carried out a number of experiments aimed at stabilising the adversarial training process, which were inspired by the literature on GANs for text generation. In particular, we looked at the effect of maximum likelihood pre-training of both the generator and discriminator, and found that it improved performance. We also reported that using (relatively) large batch sizes stabilised training and that providing the generator with rewards for partial sequences improved performance. We described two additional experiments which did not affect results, namely regularising the discriminator model and updating the generator's parameters more often than the discriminator's.

For these experiments, we measured performance by direct methods that compare the output from the simulator to real user data. We relied on two metrics, the F-score and KL-divergence. Based on these results, we selected one version of the model to compare to an identical model trained with MLE.

We compared the two simulators using direct and indirect evaluation methods. Direct evaluation favoured the MLE trained simulator, which obtained higher F-scores and lower KL-divergences with respect to test data. However, our main results related to the indirect evaluation of the two simulators. We used both to train dialogue policies, and then evaluated said policies against the GAN and MLE simulators, and an ABUS.

Unsurprisingly, for the two simulators used in training, the best performing policies on average were the ones they had trained. Against the ABUS, which should be neutral with respect to both training methods, results were very encouraging for the proposed model, with policies trained with the GAN simulator having higher success rates, both on average and when looking at the best performing policies.

7.2 Limitations and Directions for Future Work

While our results are encouraging, several aspects of this research topic could be investigated further. First, we have done no exploration of the architecture for the generator model. LSTMs seemed appropriate, and are also used by Asri et al. (2016) and Kreyssig et al. (2018), but other models may perform better, and in particular including an attention mechanism could be helpful. The number of layers and size of the hidden state could also be changed, and generally more systematic hyper-parameter tuning is necessary. Similarly, gains in performance can probably be achieved by tuning the hyper-parameters of the RL algorithm for policy optimisation.

We could also modify our model to produce predictions in natural language rather than dialogue acts. In that case, it may be interesting to explore the possibility of relying on a pre-trained transformer model, and using adversarial training for fine-tuning only, as suggested in contexts other than user simulation by Wu et al. (2021) and Croce et al. (2020). Work by Xu et al. (2018) which proposes a method to encourage GANs to produce diverse output may also be relevant, since, as we have previously discussed, output diversity is a particularly desirable property for a user simulator.

Even if focusing on the proposed model as it is, we would wish to carry out further experiments to confirm our findings. As discussed in the previous chapter, we do not rule out that the GAN simulator's performance could be achieved through maximum likelihood with appropriate regularisation. Most obviously though, the superiority of policies trained with the GAN simulator over policies trained with the MLE simulator, for which we made a case using cross-model evaluation, needs to be confirmed in interactions with real users.

References

Asri, L. E., He, J., and Suleman, K. (2016). A sequence-to-sequence model for user simulation in spoken dialogue systems. *Proceedings of the 17th Annual Conference of the International Speech Communication Association*.

Caccia, M., Caccia, L., Fedus, W., Larochelle, H., Pineau, J., and Charlin, L. (2020). Language gans falling short. In *ICLR 2020 - Proceedings of the Seventh International Conference on Learning Representation*.

Chandramohan, S., Geist, M., Lefevre, F., and Pietquin, O. (2011). User simulation in dialogue systems using inverse reinforcement learning. In *Twelfth Annual Conference of the International Speech Communication Association*.

Chandramohan, S., Geist, M., Lefevre, F., and Pietquin, O. (2012). Behavior specific user simulation in spoken dialogue systems. In *Speech Communication; 10. ITG Symposium*, pages 1–4.

Croce, D., Castellucci, G., and Basili, R. (2020). GAN-BERT: Generative adversarial learning for robust text classification with a bunch of labeled examples. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2114–2119, Online. Association for Computational Linguistics.

Crook, P. A. and Marin, A. (2017). Sequence to sequence modeling for user simulation in dialog systems. In *INTERSPEECH*.

de Masson d'Autume, C., Mohamed, S., Rosca, M., and Rae, J. (2019). Training language gans from scratch. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dhingra, B., Li, L., Li, X., Gao, J., Chen, Y.-N., Ahmed, F., and Deng, L. (2017). Towards end-to-end reinforcement learning of dialogue agents for information access. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 484–495, Vancouver, Canada. Association for Computational Linguistics.

Eckert, W., Levin, E., and Pieracini, R. (1997). User modeling for spoken dialogue system evaluation. In *Proceedings of the 1997 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 80–87. IEEE.

Eric, M., Goel, R., Paul, S., Kumar, A., Sethi, A., Ku, P., Goyal, A. K., Agarwal, S., Gao, S., and Hakkani-Tur, D. (2019). Multiwoz 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines.

Eshky, A., Allison, B., and Steedman, M. (2012). Generative goal-driven user simulation for dialog management. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 71–81. Association for Computational Linguistics.

Georgila, K., Henderson, J., and Lemon, O. (2005). Learning user simulations for information state update dialogue systems. In *Proceedings of the Ninth European Conference on Speech Communication and Technology*, pages 893–896. Ninth European Conference on Speech Communication and Technology, INTERSPEECH 2005 ; Conference date: 04-09-2005 Through 08-09-2005.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Gür, I., Hakkani-Tür, D., Tür, G., and Shah, P. (2018). User modeling for task oriented dialogues. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 900–906. IEEE.

Henderson, M., Thomson, B., and Williams, J. D. (2014). The second dialog state tracking challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 263–272, Philadelphia, PA, U.S.A. Association for Computational Linguistics.

Hou, Y., Fang, M., Che, W., and Liu, T. (2019). A corpus-free state2seq user simulator for task-oriented dialogue. In Sun, M., Huang, X., Ji, H., Liu, Z., and Liu, Y., editors, *Chinese Computational Linguistics*, pages 689–702, Cham. Springer International Publishing.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jung, S., Lee, C., Kim, K., Jeong, M., and Lee, G. G. (2009). Data-driven user simulation for automated evaluation of spoken dialog systems. *Computer Speech & Language*, 23(4):479–509.

Keizer, S., Gasic, M., Jurcicek, F., Mairesse, F., Thomson, B., Yu, K., and Young, S. (2010). Parameter estimation for agenda-based user simulation. In *Proceedings of the SIGDIAL 2010 Conference*, pages 116–123.

Keizer, S., Rossignol, S., Chandramohan, S., and Pietquin, O. (2012). *User Simulation in the Development of Statistical Spoken Dialogue Systems*, pages 39–73. Springer New York, New York, NY.

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic gradient descent. In *ICLR: International Conference on Learning Representations*, pages 1–15.

Kreyssig, F., Casanueva, I., Budzianowski, P., and Gasic, M. (2018). Neural user simulation for corpus-based policy optimisation of spoken dialogue systems. In Komatani, K., Litman, D. J., Yu, K., Cavedon, L., Nakano, M., and Papangelis, A., editors, *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue, Melbourne, Australia, July 12-14, 2018*, pages 60–69. Association for Computational Linguistics.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Li, J., Monroe, W., Shi, T., Jean, S., Ritter, A., and Jurafsky, D. (2017). Adversarial learning for neural dialogue generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2157–2169, Copenhagen, Denmark. Association for Computational Linguistics.

Li, X., Lipton, Z. C., Dhingra, B., Li, L., Gao, J., and Chen, Y.-N. (2016). A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*.

Lin, H., Lubis, N., Hu, S., van Niekerk, C., Geishauser, C., Heck, M., Feng, S., and Gašić, M. (2021). Domain-independent user simulation with transformers for task-oriented dialogue systems.

Liu, B. and Lane, I. (2017). Iterative policy learning in end-to-end trainable task-oriented neural dialog models. *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 482–489.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Nie, X., Lin, Z., Huang, X., and Zhang, Y. (2019). Graph neural net-based user simulator. In Sun, M., Huang, X., Ji, H., Liu, Z., and Liu, Y., editors, *Chinese Computational Linguistics*, pages 638–650, Cham. Springer International Publishing.

Papangelis, A., Wang, Y.-C., Molino, P., and Tur, G. (2019). Collaborative multi-agent dialogue model training via reinforcement learning. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 92–102, Stockholm, Sweden. Association for Computational Linguistics.

Peng, B., Li, X., Li, L., Gao, J., Celikyilmaz, A., Lee, S., and Wong, K.-F. (2017). Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2231–2240, Copenhagen, Denmark. Association for Computational Linguistics.

Pietquin, O. and Dutoit, T. (2006). A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(2):589–599.

Pietquin, O. and Hastie, H. (2013). A survey on metrics for the evaluation of user simulations. *The Knowledge Engineering Review*, 28(1):59–73.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Schatzmann, J., Stuttle, M. N., Weilhammer, K., and Young, S. (2005). Effects of the user model on simulation-based learning of dialogue strategies. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.*, pages 220–225. IEEE.

Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H., and Young, S. (2007). Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152, Rochester, New York. Association for Computational Linguistics.

Schatzmann, J. and Young, S. (2009). The hidden agenda user simulation model. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(4):733–747.

Scheffler, K. and Young, S. (2001). Corpus based dialogue simulation for automatic strategy learning and evaluation. In *Proceedings of the NAACL Workshop on Adaptation in Dialogue Systems*, pages 64–70.

Shi, W., Qian, K., Wang, X., and Yu, Z. (2019). How to build user simulators to train RL-based dialog systems. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1990–2000, Hong Kong, China. Association for Computational Linguistics.

Su, P.-H., Gašić, M., Mrkšić, N., Rojas-Barahona, L. M., Ultes, S., Vandyke, D., Wen, T.-H., and Young, S. (2016). On-line active reward learning for policy optimisation in spoken dialogue systems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2431–2441, Berlin, Germany. Association for Computational Linguistics.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Takanobu, R., Liang, R., and Huang, M. (2020). Multi-agent task-oriented dialog policy learning with role-aware reward decomposition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 625–638, Online. Association for Computational Linguistics.

Tseng, B.-H., Dai, Y., Kreyssig, F., and Byrne, B. (2021). Transferable dialogue systems and user simulators. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 152–166, Online. Association for Computational Linguistics.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Wu, Q., Li, L., and Yu, Z. (2021). Textgail: Generative adversarial imitation learning for text generation. In *Proceedings of the AAI Conference on Artificial Intelligence, AAAI Press*.

Xu, J., Ren, X., Lin, J., and Sun, X. (2018). Diversity-promoting GAN: A cross-entropy based generative adversarial network for diversified text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3940–3949, Brussels, Belgium. Association for Computational Linguistics.

Young, S., Gasic, M., Thomson, B., and Williams, J. (2013). Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101:1160–1179.

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.

Zhou, L., Small, K., Rokhlenko, O., and Elkan, C. (2017). End-to-end offline goal-oriented dialog policy learning via policy gradient.

Appendix A

Data

System Act	Example
welcomemsg	Hello , welcome to the Cambridge restaurant system? You can ask for restaurants by area , price range or food type . How may I help you?
offer	nandos is a nice restaurant in the south of town in the cheap price range
inform	city stop restaurant is in the north part of town .
canthelp	Sorry there is no mexican restaurant in the moderate price range
canthelp.exception	I am sorry but there is no other french restaurant that matches your request
confirm-domain	You are looking for a restaurant is that right?
reqmore	Can I help you with anything else?
impl-conf	There are restaurants serving irish food .
expl-conf	Ok , a restaurant in any part of town is that right?
repeat	Sorry I am a bit confused ; please tell me again what you are looking for .
request_food	What kind of food would you like?
request_area	What part of town do you have in mind?
request_pricerange	Would you like something in the cheap , moderate , or expensive price range?
select_food	Sorry would you like polynesian or chinese food?
select_area	Sorry would you like something in the west or in the south
select_pricerange	Sorry would you like something in the moderate price range or you dont care

Table A.1 List of system acts. Examples are copied verbatim from the DSTC-2 corpus.

User Act	Example
inform_food	thai food
inform_area	im looking for a restaurant in the south part of town
inform_this	dont care
request_food	what type of food do they serve
reqalts	is there anything else
request_addr	whats the address
request_phone	what is the phone number
request_pricerange	whats the price range
thankyou	thank you
bye	good bye
request_postcode	whats the post code
hello	hello
negate	no i want halal food
inform_pricerange	id like to find a moderately priced restaurant
request_area	what is the area
affirm	right
confirm_pricerange	is it an expensive restaurant
confirm_food	is it portuguese food
confirm_area	is it in the center of town
ack	okay um
restart	oh jesus christ start over
repeat	can you repeat that
inform_name	do you have prezzo
deny_food	not korean
deny_pricerange	uh cheap moderate
reqmore	more
deny_name	i hate golden wok is there anything else
request_name	okay what was the name
request_signature	what is their special dish

Table A.2 List of user acts. Examples are copied verbatim from the DSTC-2 corpus.

Appendix B

Additional Results

		Evaluation Simulator		
		MLE	GAN	ABUS
Training	MLE	59.32	57.23	71.08
Simulator	GAN	59.21	59.30	75.67

Table B.1 Average rewards against different simulators for policies trained with the MLE and GAN simulators. Average rewards are averaged over 3,000 evaluation dialogues, and over 10 training runs for each training simulator.

		Evaluation Simulator		
		MLE	GAN	ABUS
Training	MLE	7.632	7.774	6.860
Simulator	GAN	7.894	7.816	6.959

Table B.2 Average number of turns in a dialogue against different simulators for policies trained with the MLE and GAN simulators. Turn counts are calculated over 3,000 evaluation dialogues, and over 10 training runs for each training simulator.

	Success Rate versus		
Policy Name	MLE	GAN	ABUS
adv-0	72.0	70.9	84.9
adv-1	69.0	71.8	90.4
adv-2	74.1	73.4	83.5
adv-3	70.3	71.7	83.1
adv-4	73.4	72.5	79.5
adv-5	71.7	70.6	89.6
adv-6	71.2	71.5	83.8
adv-7	70.8	70.6	86.5
adv-8	72.5	71.9	86.4
adv-9	70.9	70.3	86.3

Table B.3 Success rates of individual policies trained against the GAN simulator. Success rates are averaged over 3,000 evaluation dialogues.

	Success Rate versus		
Policy Name	MLE	GAN	ABUS
ML-0	72.4	71.1	77.5
ML-1	70.4	68.2	71.7
ML-2	72.3	70.7	81.2
ML-3	72.3	69.6	84.1
ML-4	71.6	70.2	79.7
ML-5	71.7	70.0	80.9
ML-6	71.9	70.7	81.0
ML-7	69.9	69.1	87.4
ML-8	72.5	71.1	85.0
ML-9	72.0	69.8	76.1

Table B.4 Success rates of individual policies trained against the MLE simulator. Success rates are averaged over 3,000 evaluation dialogues.