De-aliasing States in Dialogue Modelling with Inverse Reinforcement Learning

Layla El Asri
Borealis AI[1]
layla.elasri@borealisai.com

Adam Trischler
Microsoft Research
adtrisch@microsoft.com

Geoff Gordon
Microsoft Research
ggordon@microsoft.com

Abstract

End-to-end dialogue response generation models learn dialogue state tracking, dialogue management, and natural language generation at the same time and through the same training signal. These models scale better than traditional modular architectures as they do not require much annotation. Despite significant advances, these models, often built using Recurrent Neural Networks (RNNs), exhibit deficiencies such as repetition, inconsistency, and low task-completion rates. To understand some of these issues more deeply, this paper investigates the representations learned by RNNs trained on dialogue data. We highlight the problem of state aliasing, which entails conflating two or more distinct states in the representation space. We show empirically that state aliasing often occurs when encoder-decoder RNNs are trained via maximum likelihood or policy gradients. We propose to augment the training signal with information about the future to force the latent representations of the RNNs to hold sufficient information for predicting the future. Specifically, we train encoder-decoder RNNs to predict both the next utterance as well as a feature vector that represents the expected dialogue future. We draw inspiration from the Structured-Classification Inverse Reinforcement Learning (SCIRL, Klein et al., 2012) algorithm to compute this feature vector. In experiments on a generated dataset of text-based games, the augmented training signal mitigates state aliasing and improves model performance significantly.

1 Introduction

Dialogue response generation (DRG) can be framed as a reinforcement learning problem where the actions are words or structured sequences of words. Much recent work has adopted the former framing, and has proposed data-driven training of DRG models in 2 steps: first, train the model to output responses using the maximum-likelihood objective, via teacher forcing; second, continue training with a distinct, possibly non-differentiable objective (e.g., maximizing the BLEU score) using policy-gradient methods (Papineni et al., 2002; Ranzato et al., 2016; Bahdanau et al., 2017; Strub et al., 2017; Narayan et al., 2018; Wu et al., 2018).

Encoder-decoder models based on recurrent neural networks now set the state of the art in DRG, but still exhibit deficiencies like inconsistency, poor syntax, and repetition. For example, they tend to repeat the same sentences inappropriately within a dialogue or across dialogues. This has been observed in both goal-oriented and general-purpose conversational settings (Das et al., 2017; Strub et al., 2017; Li et al., 2016; Holtzman et al., 2019). Several recent works have focused on improving the decoding mechanism of such models (Bahdanau et al., 2017; Wiseman and Rush, 2016; Wu et al., 2018; Holtzman et al., 2019). In this paper, we take the complimentary approach and investigate the representations learned by the encoder. We hypothesize that one cause for repetition is *state aliasing*, which entails conflating two or more distinct states in the representation space. Recent work has shown that an RNN encoder's hidden representations of two different dialogue contexts may be very similar if the utterances following these contexts are the same (El Asri and Trischler, 2019). In other words, when the immediate outputs are the same, the encoder may learn to represent the inputs similarly. This was demonstrated in training end-to-end dialogue models with policy gradient algorithms and we investigate whether this happens with other training settings and in particular in the maximum likelihood setting.

[1]Work done at Microsoft Research.

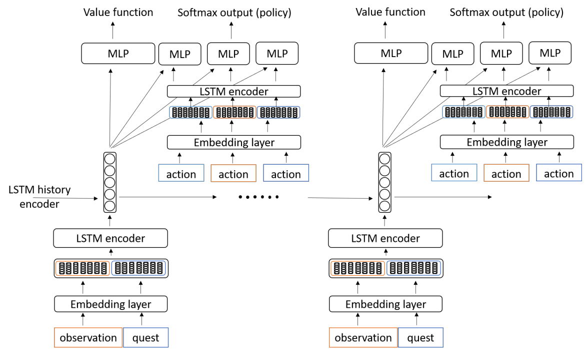

Figure 1: Retrieval-based dialogue response generation model trained on text-based games.

We train a retrieval model and a generative model with a recurrent encoder and decoder. Our experiments suggest that state aliasing also occurs in MLE training in both settings. We go on to propose a solution to state aliasing based on inverse reinforcement learning, which forces the model to learn representations that depend less on the next utterance and more on the expected future dialogue trajectory. We adapt the Structured-Classification Inverse Reinforcement Learning algorithm (SCIRL, Klein et al., 2012) to compute a feature vector representing an expectation of the dialogue future and train models to predict this feature vector. SCIRL is a simple inverse reinforcement learning algorithm with strong theoretical guarantees and which does not require knowing the transition dynamics of the environment nor having access to a simulator of the environment. With this approach, we show significant improvement on a dataset of generated text-based games.

2 Background: State Aliasing in RNNs Trained with Policy Gradient Methods

2.1 Definition

In this section, we define the state aliasing problem and summarize the main results from El Asri and Trischler (2019), which we build upon. This also serves to introduce the experimental setting and several components of the model we use in our experiments.

We consider two states s_i and s_j to be ϵ-aliased by a model M, for which $||M(s)|| \leq n \, \forall s$, when the Euclidean distance between the representations of these states in M (denoted by $M(s_i)$ and $M(s_j)$, respectively) is less than ϵ with $\epsilon < 2n$:

$$||M(s_i) - M(s_j)||_2 \leq \epsilon.$$

If ϵ is zero, the model learns exactly the same representation for both states. El Asri and Trischler (2019) studied state aliasing in neural DRG models trained with policy gradients. They showed that the model depicted in Figure 1 (which will be described below) went through phases in which different states were ϵ-aliased during training.

2.2 Experimental setting

This result was demonstrated using a proxy for dialogue response generation: playing simple text-based games constructed in TextWorld (Côté et al., 2018). Text-based games are interactive turn-based simulations that use template-based natural language to describe the game state, to accept actions from the player, and to describe consequent changes in the game environment. Thus, a text-based game can be viewed as a dialogue between the player and the environment. The game used by El Asri and Trischler (2019) to study aliasing is set in a house with several rooms. To succeed, the player must perform the following sequence of actions, called a quest: {*go west, take the blue key, go east, unlock the blue chest with the blue key, open the blue chest, take the red key, take the bottle of shampoo, go west, unlock the red chest with the red key, open the red chest, insert the bottle of shampoo into the red chest*}. Notice that to complete the game, the agent must *go west* twice. If state aliasing occurs because of this repetition, then the agent should struggle to learn to perform different actions in the two situations. The next section describes the RNN-based model trained on this game. We refer to the cited paper for more details on the experiments.

2.3 Model

At each turn, the model in Figure 1 takes as input the game-generated text observation of the state and uses its policy to select (i.e., retrieve) a textual action to take to make progress in the game. The observation describes the agent's current location (a room) and the various objects in this room. The sentences in the description are concatenated and then tokenized into words. The words are mapped to embeddings using ELMo (Peters et al., 2018),[1] and the model encodes the sequence of embeddings via LSTM (the LSTM encoder). The model encodes the quest that it must perform using a separate LSTM encoder with distinct parameters. The

[1]We use the small ELMo model available at `https://allennlp.org/elmo`.

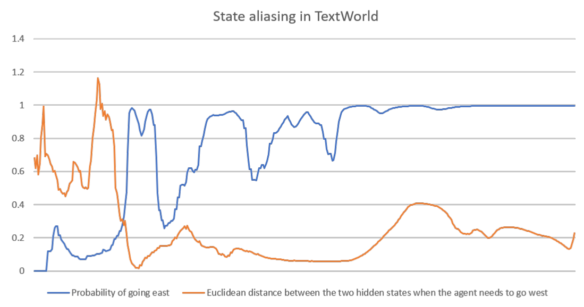

Figure 2: Illustration of state aliasing with a model trained with policy gradients. Source: El Asri and Trischler (2019)

quest consists of a short text string that describes the objective, in the form of the sequence of actions that completes the game. An example of quest is given in the appendix. The concatenation of the quest encoding and the observation encoding is passed to a higher-level LSTM called the history encoder. Its hidden state at turn t represents the game history up to that turn and gives the representation for state s_t, i.e., $M(s_t)$. At each turn, the model is provided with a set of candidate actions to select from. In the RL setting, these actions are valid actions returned by the game engine and in the MLE setting, they are randomly sampled from the list of actions present in the data. Similarly to the observations, it encodes these via LSTM. The model then replicates the history encoding and concatenates it with each action encoding. Passing these concatenated vectors separately through an MLP yields logits for a softmax function, which in turn induces a distribution $p(a_t|s_t)$ over the valid actions.

2.4 Observations

El Asri and Trischler (2019) trained their model with policy gradients to play a TextWorld game requiring the sequence of 11 actions described above. They analyzed the hidden states of the LSTM history encoder and showed that, in certain cases, the hidden representations corresponding to two different game states underwent ϵ-aliasing during training. The aliased hidden states are precisely those preceding the *go west* actions. As shown on Figure 2, the Euclidean distance between these two states decreases during training. When the distance reaches as low as 0.002, the agent learns to go east when it should have instead inserted the shampoo bottle into the chest. In other words, when the agent goes west again, its hidden state looks very similar to the one when it went west for the first time, and this propagates to subsequent hidden states such that the agent repeats other former actions that came after going west (e.g., going east). This impedes the agent's training and the agent often never recovers from this aliasing.

The intuition given for this phenomenon is that the output distribution at separate states with the same optimal action looks very similar (i.e., close to 1 for the optimal action and 0 for the others). Policy gradients then push the corresponding hidden states together. Experiments suggest that entropy-based regularization helps mitigate this issue. Adding an entropy-based bonus to the loss function forces the output distribution to be less peaked, so even if the optimal action is the same for two different states, the policy distribution might differ enough to represent the states differently. Another helpful modification is to train the RL agent to output not only the policy (as a distribution over actions) but also a baseline function representing the expected sum of rewards at each state. If states share the same optimal actions but not the same expected sum of rewards, then the model is forced to learn different representations to predict the baseline accurately at each state.

It was left as an open question whether the same sort of aliasing occurs with other training objectives such as classification by maximum likelihood. Below, we investigate this question.

3 A Proxy Dataset for Dialogue

For our experiments we use a dataset of text-based games built with TextWorld (Côté et al., 2018). All games share the same environment layout and

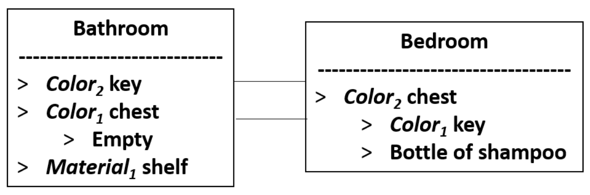

Figure 3: Structure of the text-based games used in experiments.

overarching structure, depicted in Figure 3. We define 10 different quests with lengths between 1 and 11 actions. There is only 1 quest of length 11 and it is similar to the game proposed by El Asri and Trischler (2019). The other quests are either subquests of this trajectory or contain slight variations.

We generate the training set by building games for all 10 quests with 200 different combinations of ($Color_1$, $Color_2$, $Material_1$). We define 5 different colors (so 20 combinations of $Color_1$, $Color_2$) and 10 different materials. This process yields 2000 games. The training data consist of the optimal trajectory for each game, including the description of the quest and the observations returned by the game engine for the shortest sequence of actions that completes the quest. We build validation and test sets by the same process, but with distinct colors and shelf materials. We define 5 unseen colors and 5 unseen materials for the validation set and likewise for the test set. This yields 1000 trajectories each for validation and test. We will make the generation code and dataset available upon publication. An example trajectory as well as the list of quests and the lists of colors and materials are given in the appendix.

We designed this dataset as a simplified proxy for short, goal-oriented dialogues. The aim is to require behaviors analogous to those of a dialogue agent. Such an agent must learn to understand, recall, and make use of many distinct entities referenced by the user, and to adapt to the different ways users express themselves; however, there are clear patterns to dialogue types (e.g., planning travel) and sub-parts (e.g., booking a flight, renting a car). Similarly, TextWorld provides a common game structure with many distinct entities and combinations. Just like a dialogue system must remember the entities mentioned by the user during the dialogue, an agent successful on our dataset must remember the objects that it has collected (e.g., it is necessary to first get a key to unlock a chest). Although the TextWorld engine produces templated language, it exhibits a degree of variability. For instance, given the same quest, the engine may emit *"Welcome to TextWorld! Here is how to play! First thing I need you to do is to head west..."*, or *"It's time to explore the amazing world of TextWorld! Here is how to play! First stop, try to venture west..."*, or some other slight variant as the observable description.

We designed the quests so that in the longest ones, the agent needs to take the same action of going west twice. Other actions share similar first tokens, e.g., *go west* and *go east*, *open* $Color_1$ *chest* and *open* $Color_2$ *chest*, etc. This dataset is thus specifically designed to study whether state aliasing occurs in similar conditions in max likelihood training and explore solutions that solve this problem. In the next section, we describe the different models trained with max likelihood and our experimental observations on state aliasing in this setting.

4 State Aliasing during Maximum Likelihood Training

4.1 Models

4.1.1 Retrieval Models

This model was described in Section 2. Note that the model we train here does not include the value function head in Figure 1 since this is only used for RL training.

We optimize this model on the training set $\mathcal{D}$ described in Section 3 with the following loss function:

$$\mathcal{L}_{\text{ret}} = -\frac{1}{|\mathcal{D}|} \sum_{s_t, a_t \in \mathcal{D}} \log p_\theta(a_t | s_t), \qquad (1)$$

where θ are the model parameters and $p_\theta(a_t|s_t)$ is the probability assigned to the correct action a_t from among the candidate actions a_j in state s_t.

We also train a version of this model that uses the following attention scheme α, which attends over history encodings h_τ based on the action encodings:

$$\alpha_{\tau j} = \text{softmax}(h_\tau^T \overline{a}_j)$$
$$\tilde{h}_j = \sum_\tau \alpha_{\tau j} h_\tau,$$

where h_τ is the hidden state at turn $\tau \leq t$ of the LSTM history encoder and $\overline{a}_j$ is the encoding vector of jth candidate action at turn t. We concatenate the derived hidden states $\tilde{h}_j$ with the respective action encodings, then pass these through the MLP and softmax that induce the policy distribution.

4.1.2 Generation Model

To investigate response generation rather than retrieval, we replace the MLP+softmax decoder of the previous model, which selects from a given set of commands, with an LSTM decoder that generates each command sequentially, word-by-word. The LSTM decoder takes as input the previously generated token (after it has been mapped to its corresponding ELMo embedding); its hidden state is initialized with that of the history encoder. To generate an output token, we take the product of the decoder's hidden state at the current time step and an output embedding matrix, then pass the result through a softmax. Our best results came from using the output vocabulary's corresponding ELMo embeddings as the output matrix, inspired by Press and Wolf (2016). As in the retrieval setup, we also train a version of this sequence-to-sequence model that uses attention. In this case, the decoder uses its current hidden representation to attend over the history encoder's hidden states in the usual way (Bahdanau et al., 2014).

Our generation models are trained using teacher-forcing to minimize the negative log-likelihood of the utterances observed in $\mathcal{D}$: for each action utterance in the training set, we sum the log-likelihoods under the decoder model for each token in that utterance, where the input to the decoder is the groundtruth previous token for the given time step. The loss function is

$$\mathcal{L}_{\text{gen}} = -\frac{1}{|\mathcal{D}|} \sum_{s_t, a_t \in \mathcal{D}} \sum_{w_i \in a_t} \log p_\theta(w_i | w_{i-1}, s_t), \tag{2}$$

where θ are the model parameters and $p_\theta(w_i|w_{i-1}, s_t)$ is the probability of the ith token in the action utterance a_t taken in state s_t. The previous token w_{i-1} comes from the groundtruth sequence. We construct the model's history encoding, which represents state s_t as $M(s_t)$, by processing the groundtruth trajectory up to the current turn in the game. Recall that this is a sequence of textual observations of the evolving game state; we process it through the two LSTM encoders detailed in Section 2. During inference, we use beam search to generate the next utterance.

Implementation details for the retrieval and generative models are given in the appendix.

4.2 Results

In Table 1, we report the accuracy of the different models in terms of exact match on the test set. The exact match score increments by 1 if the retrieved or generated action matches the ground truth action word for word. This is not the best way to evaluate DRG but it makes sense in the case of TextWorld games since the parser only accepts a limited set of sentences. To relax the evaluation, we also compute *action precision* on the test set: this is defined similarly to the exact match metric, except that we we ignore color and material adjectives. Action precision thus measures only whether commands are correct in terms of verbs and objects.

Model	Accuracy	Action Precision
Retrieval	0.612	0.675
Retrieval + attention	0.711	0.801
Generation + attention	0.262	0.478

Table 1: Accuracy and action precision of the different models trained on text-based games.

4.2.1 State Aliasing

To identify state aliasing in models, we inspect the Euclidean distance between hidden states for certain examples from the test set. Let us consider the following quest: {*go west, take Color*$_2$ *key, go east, unlock Color*$_2$ *chest with the Color*$_2$ *key, open Color*$_2$ *chest, take Color*$_1$ *key, take the bottle of shampoo, go west, unlock Color*$_1$ *chest with Color*$_1$ *key, open Color*$_1$ *chest, insert the bottle of shampoo into Color*$_1$ *chest*}. Several actions are repeated, namely going west, unlocking a chest, and opening a chest. We look at the Euclidean distance between hidden states when the LSTM history encoder processes this trajectory up to the point where it must perform the last action. This game is similar to the one used by El Asri and Trischler (2019) to study state aliasing in the RL setting. They observed that states often became aliased after the agent should go west for the second time.

With the retrieval model, we observe similar behaviour: whenever the action of going east or going west is available to the agent at the end of the game, it consistently chooses this incorrect action over all others, with a probability close to 1. Focusing on these failure cases, we show in Figure 4 that states s_E, s_W, which correspond to going east and going west for the second time, respectively, are consistently the closest in Euclidean space, with an average distance of 0.00013. The average distance from s_W to all other previous hidden states is 0.060. Thus, for all trajectories where the agent incorrectly goes east or west at the end of the game, s_E is the closest hidden state to s_W by several orders of magnitude. Following states s_E and s_W, the agent must unlock and open a chest. Since the agent must repeat the same sequence of actions, it represents the immediately previous states similarly. This aliasing has downstream effects on

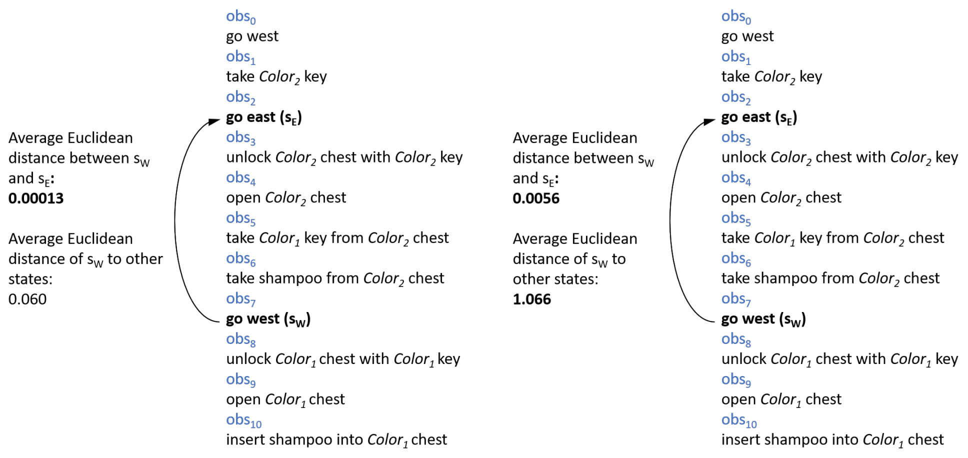
Average Euclidean distance between s_W and s_E:
0.00013
Average Euclidean distance of s_W to other states:
0.060
obs$_0$
go west
obs$_1$
take *Color$_2$* key
obs$_2$
go east (s_E)
obs$_3$
unlock *Color$_2$* chest with *Color$_2$* key
obs$_4$
open *Color$_2$* chest
obs$_5$
take *Color$_1$* key from *Color$_2$* chest
obs$_6$
take shampoo from *Color$_2$* chest
obs$_7$
go west (s_W)
obs$_8$
unlock *Color$_1$* chest with *Color$_1$* key
obs$_9$
open *Color$_1$* chest
obs$_{10}$
insert shampoo into *Color$_1$* chest
Average Euclidean distance between s_W and s_E:
0.0056
Average Euclidean distance of s_W to other states:
1.066
obs$_0$
go west
obs$_1$
take *Color$_2$* key
obs$_2$
go east (s_E)
obs$_3$
unlock *Color$_2$* chest with *Color$_2$* key
obs$_4$
open *Color$_2$* chest
obs$_5$
take *Color$_1$* key from *Color$_2$* chest
obs$_6$
take shampoo from *Color$_2$* chest
obs$_7$
go west (s_W)
obs$_8$
unlock *Color$_1$* chest with *Color$_1$* key
obs$_9$
open *Color$_1$* chest
obs$_{10}$
insert shampoo into *Color$_1$* chest

Figure 4: Illustration of state aliasing after max likelihood training. Left: retrieval model without attention, right: retrieval model with attention.

the model's trajectory, leading to its failure at the final game state: it elects to go west or to go east rather than placing the bottle. Go actions occur after unlocking and opening the earlier chest, and aliasing locks the model into looping behaviors.

We observe the same phenomenon in the retrieval model with attention: s_W is always closest to s_E in Euclidean distance. On average, this distance is 0.0056; the average distance to other states is 1.066. In addition, we observe the same failure mode: whenever the agent has the choice to go east or go west at the end, it does so.

Interestingly, the generation model's performance suffers differently: the model tends to produce the *go west* action much more frequently, including when it should produce the {take *Color*$_2$ key} or the {open *Color*$_2$ chest} actions. We suspect that the second occurrence is due to state aliasing with the first going west action. As for the first occurrence, this seems to suggest that encoder-decoder RNN models are prone to state aliasing based on temporal distance (the agent goes west and then repeats the action of going west). In general, the behaviors of this more complicated model are harder to interpret but show symptoms of state aliasing.

5 Augmenting the Training Signal with Future Feature Vectors

5.1 Motivation

State aliasing is a problem in partially-observable environments: agents with limited sensors might not always be able to differentiate distinct states. This problem has therefore been studied extensively. In our case, states are aliased in the hidden space rather than the sensor space, but we can apply ideas from the partially-observable literature to mitigate the problem. (McCallum, 1996) proposed to learn a representation of an RL agent's state based on its expected sum of rewards. In the deep reinforcement learning literature, this corresponds to approaches like DQN (Mnih et al., 2013), where the RL agent's representation is based on the estimated Q-values or on the estimated values of states given by models trained to predict the next best action. In this last setting, experiments by (El Asri and Trischler, 2019) suggest that training an RNN model subject to state aliasing to predict states' values (in addition to the next action) helps to prevent aliasing.

We propose a related approach for the maximum likelihood setting. Dialogue datasets often do not come with ratings, so there is no natural reward function for training a dialogue agent. However, we can adapt algorithms from the inverse reinforcement learning literature to learn a reward function from data, assuming that the data consists of expert trajectories. In particular, the Structured Classification Inverse Reinforcement Learning (SCIRL, Klein et al., 2012) algorithm enables learning a reward function without knowing the true environment model nor having access to an environment simulator. We describe this algorithm in the next section.

```
Algorithm 1: Expert feature expectation estimation
input : A dataset of history-response pairs $\mathcal{D} = \{(s_i, a_i = \pi_E(s_i))_{1\leq i\leq N}\}$
output: An estimation $\hat{\mu}^{\pi_E}(s)$ of the expert feature expectation $\mu^{\pi_E}(s)$ at each state $s$
1 Let $s_t$ be dialogue history at turn $t$. Compute the feature vector $\phi(s_t) = (\phi_1(s_t), ..., \phi_p(s_t))$ as
    $\phi(s) = \frac{1}{t}\sum_{t'\leq t} \text{emb}(s_{t'})$;
2 Compute $\hat{\mu}^{\pi_E}(s)$ through SARSA updates on $\mathcal{D}$ such that $\forall\ i, \hat{\mu_i}^{\pi_E}(s)$ is the value function for the
    reward $\phi_i(s)$ and the expert policy $\pi_E$.
```

5.2 SCIRL

SCIRL relies on a linear parameterization of the reward function $R_\theta = \theta^T\phi(s)$ where θ are parameters and $\phi(s) = (\phi_1(s), ..., \phi_p(s))^T$ is a feature vector of p basis functions ϕ_i. Let us define a policy π as a mapping from a state-action pair (s, a) to a probability $\pi(s, a) \in [0, 1]$ and a greedy policy π as a mapping from states to actions: $\pi(s) = a$. Given this parameterization, the Q function for a policy π evaluated with this reward function is $Q^\pi_\theta(s, a) = \theta^T\mu^\pi(s, a)$, where $\mu^\pi(s, a)$ is the *feature expectation* vector: $\mu^\pi(s, a) = \mathbb{E}[\sum_{t\geq 0}\gamma^t\phi(s_t) \mid S_0 = s, A_0 = a, \pi]$.

The SCIRL algorithm proceeds as follows. Suppose we have a dataset $\mathcal{D}$ of state action pairs collected with a greedy expert policy π_E: $\mathcal{D} = \{(s_i, a_i = \pi_E(s_i))_{1\leq i\leq N}\}$. In our case, states are dialogue histories and actions are dialogue responses. These pairs are collected from human conversations or, in TextWorld, by generating game trajectories. Suppose that we also have an estimate of the expert feature expectation vector μ^{π_E} and access to a multi-class classification (MC^2 for short) algorithm. We use this MC^2 algorithm to learn a decision rule $g_{\theta_c}(s) = a = \text{argmax}_{a'}\,\theta_c^T\mu^{\pi_E}(s, a')\ \forall\ s, a \in \mathcal{D}$. In other words, we learn parameters θ_c such that the actions in the dataset are the optimal actions given the score function $\theta_c^T\mu^{\pi_E}(s, a)$. The reward function $R_{\theta_c}(s) = \theta_c^T\mu^{\pi_E}(s, a)$ is then a reward function under which the greedy expert policy π_E is optimal; i.e., this policy maximizes the expected sum of rewards for all states s.

5.3 Using Expert Feature Expectations to Train Dialogue Models

In the case of dialogue, we showed that states become aliased when the outputs for those states are similar. We could compute a reward as in SCIRL, then use this to compute expected sums of rewards at each state, then predict the expected value using our model. However, we only need an indication of which states should be differentiated. The expert feature expectation vector is sufficient for this purpose. Indeed, this vector captures the sum of features that will be observed after visiting a state. If states lead to different outcomes, then their feature expectations should be different. The combination of the expectation vector and the parameters θ_c gives an extra indication of which actions are optimal, but because we are training with MLE rather than RL, the optimal actions should be learned through the MLE objective. We thus propose only to estimate the expert feature expectation vector and have our model predict this quantity at each state. We define the expert feature expectation for a state s as $\mu^{\pi_E}(s) = \mathbb{E}[\sum_{t\geq 0}\gamma^t\phi(s_t) \mid S_0 = s, \pi_E]$. This approach relates to the predictive state representation literature where states are built to hold sufficient information to predict the future (Downey et al., 2017).

Klein et al. (2012) observed that $\forall i,\ \mu_i^{\pi_E}(s)$ corresponds to the value function of π_E evaluated with the reward function ϕ_i: $\mu_i^{\pi_E}(s) = V^\pi_{\phi_i}(s) = \mathbb{E}[\sum_{t\geq 0}\gamma^t\phi_i(s) \mid S_0 = s, \pi_E]$. We will use this observation to compute an estimate of the expert feature expectation vector. This approach is described in Algorithm 1. The first step consists of computing the feature vector ϕ, as the SCIRL algorithm assumes ϕ is given. This vector must be a meaningful representation of the state, which is in our case the dialogue history. We use a very simple representation that averages the embeddings of all past observations: the feature vector $\phi(s)$ for state s is the average sentence embedding for all sentences preceding time t, while the sentence embeddings are the average of a sentence's word embeddings. This representation of dialogue history only requires access to pre-trained word embeddings. The second step of the algorithm computes the estimates $\hat{\mu}^{\pi_E}(s)$. For this, we compute the value functions $V^{\pi_E}_{\phi_i}$ for all i. We use the SARSA algorithm (Rummery and Niranjan,

1994) to compute these estimates. We train a model $M^S_{\theta_s}$ with the following loss function: $\mathcal{L} = \sum_{s,s' \in \mathcal{D}} \left(\sum_i \left(\phi_i(s) + \gamma M^S_{\theta_s}(s') - M^S_{\theta_s}(s) \right)^2 \right)$.

SARSA updates are equivalent to Q-learning updates, except that they estimate the value of the policy used for data collection instead of estimating the optimal policy. We define our estimator $\hat{\mu}^{\pi_E}(s) = M^S_{\theta_s}(s) \; \forall \; s$.

We modify our retrieval and generative models to incorporate a head that predicts the output of M^S at each state s. We change the learning objectives of these models by adding the mean squared error loss,

$$\mathcal{L}_{\text{sar}} = \frac{1}{D} \sum_{s \in \mathcal{D}} \left(M^S(s) - M(s) \right)^2 , \qquad (3)$$

where M indicates either the generative or the retrieval model. We combined losses as follows: $\mathcal{L} = \lambda_{MLE} \, \mathcal{L}_{ret} + \lambda_{sar} \, \mathcal{L}_{sar}$ and $\mathcal{L} = \lambda_{MLE} \, \mathcal{L}_{gen} + \lambda_{sar} \, \mathcal{L}_{sar}$ for the retrieval and generative models, respectively. We achieved the best results with $\lambda_{MLE} = 0.1$ and $\mathcal{L}_{sar} = 1000$ in both cases. Further implementation details are given in the appendix.

6 Results

Results on the test set are given in Table 2. A first observation is that adding expert feature expectations improves the performance significantly for both the retrieval and the generative models. In the retrieval case, accuracy improves by more than 20% and action precision improves by 18.6%. Notably, all the instances of state aliasing illustrated on Figure 4 were resolved by this solution: when the augmented model had the possibility of going east at the end of the longest games, for instance, it never chose this option.

Adding feature expectations improves the accuracy of the generative model by 8% and its action precision by 49%. The high action precision suggests that low accuracy is a result of poor generalization to unseen entities. To counter this, we added a copy mechanism similar to that of (Manning and Eric, 2017). This mechanism enables the model either to generate a word from the vocabulary (of size 53 in our dataset) or to point to a word in the dialogue history. Similarly to (Manning and Eric, 2017), we only allow the model to copy entities. In a goal-oriented dialogue dataset, entities are given by a knowledge base. In our case, we assume that we have a knowledge base of game entities, i.e., chest colors and shelf materials. We also limit copying to the last dialogue turn, which is sufficient since the TextWorld observations always describe all the elements present in the room where the agent is currently located. Adding this copying mechanism improved accuracy by 46% while maintaining a high action precision. This confirms that the generation model's low accuracy results mostly from poor generalization to unseen entities. Note that combining the attention schemes with the expert feature expectation prediction did not yield the best results in our experiments. We observed that in this case, even though hidden states were de-aliased, attention states caused aliasing and an increased number of failures.

Our results are encouraging and show how state aliasing can damage the performance of a model. In both generative and retrieval cases, gains of more than 18% in action precision are made by addressing state aliasing through estimation of expert feature expectations.

Model	Accuracy	Action Precision
Ret	0.612	0.675
Ret + attn	0.711	0.801
Ret + feat exp	**0.915**	**0.987**
Ret + attn + feat exp	0.733	0.801
Gen + attn	0.262	0.478
Gen + feat exp	0.342	**0.967**
Gen + attn + feat exp	0.273	0.368
Gen + feat exp + copy	**0.799**	0.828
Gen + attn + copy	0.465	0.481

Table 2: Accuracy and action precision of the different models trained on text-based games. *Ret* is the retrieval model and *Gen* is the generative model. *feat exp* stands for feature expectations and *attn* stands for attention.

7 Conclusion

This work investigated state aliasing in RNNs in the maximum likelihood setting, building on related work in the RL setting. We highlighted that RNNs are prone to aliasing states that share similar optimal actions. We augmented the maximum likelihood objective with a loss that encourages the model to represent states based on expected dialogue futures. We showed that this augmented objective mitigates state aliasing and significantly improves performance on a dataset of generated text-based games.

References

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2017. An Actor-Critic Algorithm for Sequence Prediction. In *Proceedings of the International Conference on Learning Representations*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben A. Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew J. Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2018. Textworld: A learning environment for text-based games. In *Proceedings of the Computer Games Workshop at ICML/IJCAI*.

Abhishek Das, Satwik Kottur, José M. F. Moura, Stefan Lee, and Dhruv Batra. 2017. Learning Cooperative Visual Dialog Agents with Deep Reinforcement Learning. In *Proceedings of the International Conference on Computer Vision*.

Carlton Downey, Ahmed Hefny, Byron Boots, Geoffrey J Gordon, and Boyue Li. 2017. Predictive state recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 6053–6064.

Layla El Asri and Adam Trischler. 2019. A study of state aliasing in RNNs with structured prediction. In *Deep reinforcement learning meets structured prediction workshop at ICLR*.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *CoRR*, abs/1904.09751.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Edouard Klein, Matthieu Geist, Bilal Piot, and Olivier Pietquin. 2012. Inverse reinforcement learning through structured classification. In *Advances in Neural Information Processing Systems*.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep Reinforcement Learning for Dialogue Generation. In *Proceeding of the Conference on Empirical Methods on Natural Language Processing*.

Christopher D. Manning and Mihail Eric. 2017. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics*.

Andrew Kachites McCallum. 1996. *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. thesis.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing atari with deep reinforcement learning. In *Proceedings of the NeurIPS workshop on Deep Reinforcement Learning*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Ranking sentences for extractive summarization with reinforcement learning. In *Proceedings of the North American Chapter of the Annual Conference of the Association for Computational Linguistics*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the Annual Conference of the Association for Computational Linguistics*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the North American Chapter of the Annual Conference of the Association for Computational Linguistics*.

Ofir Press and Lior Wolf. 2016. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence Level Training with Recurrent Neural Networks. In *Proceedings of the International Conference on Learning Representations*.

G. A. Rummery and M. Niranjan. 1994. On-Line Q-Learning Using Connectionist Systems. Technical report, Cambridge University.

Florian Strub, Harm de Vries, Jeremie Mary, Bilal Piot, Aaron Courville, and Olivier Pietquin. 2017. End-to-end optimization of goal-driven and visually grounded dialogue systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

L. Wu, F. Tian, T. Qin, J. Lai, and T.-Y. Liu. 2018. A study of reinforcement learning for neural machine translation. In *Proceeding of the Conference on Empirical Methods on Natural Language Processing*.

A Implementation Details

Retrieval and generative models The ELMo embedding layer outputs vectors of size 256 for each word. The LSTM encoder, the LSTM history encoder, and the MLP use hidden states of size 128. The LSTM encoder and the LSTM history encoder both have one layer of LSTM units. The MLP that preceded the softmax operator uses $\tanh$ activation in its hidden layer. We use Adam (Kingma and Ba, 2014) with a learning rate of 0.002 and we clip the gradient norm to 10. The retrieval model is given 10 candidate actions to select from. These actions are uniformly sampled from the set of all possible actions.

The decoder of the generative model uses hidden states of size 256 and outputs words which belong to a vocabulary of size 53. Beam search is performed with a beam of width 4. We use Adam (Kingma and Ba, 2014) with a learning rate of 0.0001 and we clip the gradient norm to 10 for both encoder and decoder. All our models are trained on batches of size 32.

Expert feature expectation model We used ELMo embeddings of size 256 to compute the feature vector ϕ. The model M^S is a multi-layer perceptron with two hidden layers of size 256 and $\tanh$ activation. We trained this model on the TextWorld dataset with batches of size 64. We used stochastic gradient descent with a learning rate of 1.10^{-7}. We chose model parameters through early stopping.

B Example of trajectory in the TextWorld dataset

Quest I hope you're ready to go into rooms and interact with objects, because you've just entered TextWorld! Your first objective is to make an effort to travel west. After that, take the red key from the Bathroom. Then, head east. Then, Insert the red key into the red chest's lock to unlock it. And then, ensure that the red chest is open. And then, take the blue key from the red chest within the Bedroom. After that, recover the shampoo from the red chest inside the Bedroom. After that, go west. After that, make sure that the blue chest in the Bathroom is unlocked with the blue key. And then, make it so that the blue chest is open. After that, you can put the shampoo inside the blue chest within the Bathroom. And if you do that, you're the winner!

Trajectory -= Bedroom =- You've just walked into a Bedroom. You can make out a red chest. There is an unguarded exit to the west.

go west

-= Bathroom =- You find yourself in a Bathroom. An ordinary kind of place. You can make out a locked usual looking blue chest in the corner. You make out a wooden shelf. The wooden shelf is typical. Unfortunately, there isn't a thing on it. There is an exit to the east. Don't worry, it is unblocked. There is a red key on the floor.

take red key

-= Bathroom =- You find yourself in a Bathroom. An ordinary kind of place. You can make out a locked usual looking blue chest in the corner. You make out a wooden shelf. The wooden shelf is typical. Unfortunately, there isn't a thing on it. There is an exit to the east. Don't worry, it is unblocked.

go east

-= Bedroom =- You've just walked into a Bedroom. You can make out a red chest. There is an unguarded exit to the west.

unlock red chest with red key

-= Bedroom =- You've just walked into a Bedroom. You can make out a red chest. There is an unguarded exit to the west.

open red chest

-= Bedroom =- You've just walked into a Bedroom. You can make out a red chest. The red chest contains a blue key and a shampoo. There is an unguarded exit to the west.

take blue key from red chest

-= Bedroom =- You've just walked into a Bedroom. You can make out a red chest. The red chest contains a shampoo. There is an unguarded exit to the west.

take shampoo from red chest

-= Bedroom =- You've just walked into a Bedroom. You can make out a red chest. The red chest is empty! This is the worst thing that could possibly happen, ever! There is an unguarded exit to the west.

go west

-= Bathroom =- You find yourself in a Bathroom. An ordinary kind of place. You can make out a locked usual looking blue chest in the corner. You make out a wooden shelf. The wooden shelf is typical. Unfortunately, there isn't a thing on it. There is an exit to the east. Don't worry, it is unblocked.

unlock blue chest with blue key

-= Bathroom =- You find yourself in a Bathroom.

An ordinary kind of place. You can make out a closed usual looking blue chest in the corner. You make out a wooden shelf. The wooden shelf is typical. Unfortunately, there isn't a thing on it. There is an exit to the east. Don't worry, it is unblocked.
open blue chest
-= Bathroom =- You find yourself in a Bathroom. An ordinary kind of place. You can make out an opened usual looking blue chest in the corner. The blue chest is empty! What a waste of a day! You make out a wooden shelf. The wooden shelf is typical. Unfortunately, there isn't a thing on it. There is an exit to the east. Don't worry, it is unblocked.
insert shampoo into blue chest

C Details of the TextWorld dataset

C.1 List of quests

- go west, take $Color_2$ key, go east, unlock $Color_2$ chest with $Color_2$ key, open $Color_2$ chest, take $Color_1$ key from $Color_2$ chest, take shampoo from $Color_2$ chest, go west, unlock $Color_1$ chest with $Color_1$ key, insert shampoo into $Color_1$ chest

- go west, take $Color_2$ key, go east, unlock $Color_2$ chest with $Color_2$ key

- go west, take $Color_2$ key, go east, unlock $Color_2$ chest with $Color_2$ key, open $Color_2$ chest, take $Color_1$ key from $Color_2$ chest, go west, unlock $Color_1$ chest with $Color_1$ key

- go west, take $Color_2$ key, go east, unlock $Color_2$ chest with $Color_2$ key, open $Color_2$ chest, take shampoo from $Color_2$ chest

- go west, take $Color_2$ key, go east, unlock $Color_2$ chest with $Color_2$ key, open $Color_2$ chest, take shampoo from $Color_2$ chest, go west, put shampoo on $Material_1$ shelf

- go west

- go west, take $Color_2$ key, go east, unlock $Color_2$ chest with $Color_2$ key, open $Color_2$ chest, take $Color_1$ key from $Color_2$ chest

- go west, take $Color_2$ key, go east, unlock $Color_2$ chest with $Color_2$ key, open $Color_2$ chest, take $Color_1$ key from $Color_2$ chest, go west, unlock $Color_1$ chest with $Color_1$ key, insert $Color_2$ key into $Color_1$ chest

- go west, take $Color_2$ key, go east, unlock $Color_2$ chest with $Color_2$ key, open $Color_2$ chest, take $Color_1$ key from $Color_2$ chest, go west, unlock $Color_1$ chest with $Color_1$ key, insert $Color_1$ key into $Color_1$ chest

- go west, take $Color_2$ key, go east, unlock $Color_2$ chest with $Color_2$ key, open $Color_2$ chest, take $Color_1$ key from $Color_2$ chest, go west, put $Color_1$ key on $Material_1$ shelf

C.2 List of entities

Training Data
Colors: blue, red, brown, white, black
Materials: wooden, gold, silver, bronze, copper, marble, iron, platinum, oak, ebony

Validation Data
Colors: purple, pink, orange, cyan, ochre
Materials: plastic, fabric, cardboard, brick, stone

Test Data
Colors: grey, yellow, vermilion, crimson, green
Materials: metal, clay, ceramic, paper, diamond