Feudal Dialogue Management with Jointly Learned Feature Extractors

Iñigo Casanueva*, **Paweł Budzianowski, Florian Kreyssig,**
Stefan Ultes, Bo-Hsiang Tseng, Yen-chen Wu and Milica Gašić
Department of Engineering, University of Cambridge, UK
{ic340,pfb30,mg436}@cam.ac.uk

Abstract

Reinforcement learning (RL) is a promising dialogue policy optimisation approach, but traditional RL algorithms fail to scale to large domains. Recently, Feudal Dialogue Management (FDM), has shown to increase the scalability to large domains by decomposing the dialogue management decision into two steps, making use of the domain ontology to abstract the dialogue state in each step. In order to abstract the state space, however, previous work on FDM relies on handcrafted feature functions. In this work, we show that these feature functions can be learned jointly with the policy model while obtaining similar performance, even outperforming the handcrafted features in several environments and domains.

1 Introduction

In task-oriented Spoken Dialogue Systems (SDS), the Dialogue Manager (DM) (or policy) is the module in charge of deciding the next action in each dialogue turn. One of the most popular approaches to model the DM is Reinforcement Learning (RL) (Sutton and Barto, 1999), having been studied for several years (Levin et al., 1998; Williams and Young, 2007; Henderson et al., 2008; Pietquin et al., 2011; Gašić et al., 2013; Young et al., 2013). However, as the dialogue state space increases, the number of possible trajectories needed to be explored grows exponentially, making traditional RL methods not scalable to large domains.

Recently, Feudal Dialogue Management (FDM) (Casanueva et al., 2018) has shown to increase the scalability to large domains. This approach is based on Feudal RL (Dayan and Hinton, 1993), a hierarchical RL method that divides a task spatially rather than temporally, decomposing the decisions into several steps and using different levels of abstraction for each sub-decision. When applied to domains with large state and action spaces, FDM showed an impressive performance increase compared to traditional RL policies.

However, the method presented in Casanueva et al. (2018), named FDQN[1], relied on handcrafted feature functions in order to abstract the state space. These functions, named Domain Independent Parametrisation (DIP) (Wang et al., 2015), are used to transform the belief of each slot into a fixed size representation using a large set of rules.

In this paper, we demonstrate that the feature functions needed to abstract the belief state in each sub-decision can be jointly learned with the policy. We introduce two methods to do it, based on feed forward neural networks and recurrent neural networks respectively. A modification of the original FDQN architecture is also introduced which stabilizes learning, avoiding overfitting of the policy to a single action. Policies with jointly learned feature functions achieve similar performance to those using handcrafted ones, with superior performance in several environments and domains.

2 Background

Dialogue management can be cast as a continuous MDP (Young et al., 2013) composed of a finite set of actions $\mathcal{A}$, a continuous multivariate belief state space $\mathcal{B}$ and a reward function $\mathcal{R}(b_t, a_t)$. At a given time t, the agent observes the belief state $b_t \in \mathcal{B}$, executes an action $a_t \in \mathcal{A}$ and receives a reward $r_t \in \mathbb{R}$ drawn from $\mathcal{R}(b_t, a_t)$. The action taken, a, is decided by the *policy*, defined as the function $\pi(b) = a$. The objective of RL is to find the optimal policy π^* that maximizes

*Currently at PolyAI, inigo@poly-ai.com

[1] In the rest of the paper we will refer to the FDM model presented in Casanueva et al. (2018) as FDQN.

Proceedings of the SIGDIAL 2018 Conference, pages 332–337,
Melbourne, Australia, 12-14 July 2018.

the expected return R in each belief state, where $R = \sum_{\tau=t}^{T-1} \gamma^{(\tau-t)} r_\tau$, γ is a discount factor, t is the current timestep and T is the terminal timestep.

There are 2 major approaches to model the policy, *Policy-based* and *Value-based* algorithms. In the former, the policy is directly parametrised by a function $\pi(b;\theta) = a$, where θ are the parameters learned in order to maximise R. In the later, the optimal policy can be found by greedily taking the action which maximises the *Q-value*, $Q^\pi(b,a)$, defined as the expected R, starting from state b, taking action a, and then following policy π until the end of the dialogue at time step T:

$$Q^\pi(b,a) = \mathbb{E}\{R|b_t = b, a_t = a\} \quad (1)$$

2.1 Feudal Dialogue Management

In FDM (Casanueva et al., 2018) (Fig. 1), the (summary) actions are divided in two subsets; slot independent actions $\mathcal{A}_i$ (e.g. hello(), inform()); and slot dependent actions $\mathcal{A}_d$ (e.g. request(), confirm()). In addition, a set of master actions $\mathcal{A}_m = (a_i^m, a_d^m)$ is defined, where a_i^m corresponds to taking an action from $\mathcal{A}_i$ and a_d^m to taking an action from $\mathcal{A}_d$. The feudal dialogue policy, $\pi(b) = a$, decomposes the decision in each turn into two steps. In the first step, the policy decides to take either a slot independent or a slot dependent action. In the second step, the state of each sub-policy is abstracted to account for features related to that slot, and a primitive action is chosen from the previously selected subset. In order to abstract the dialogue state for each sub-policy, a feature function $\phi_s(b) = b_s$ is defined for each slot $s \in \mathcal{S}$, as well as a slot independent feature function $\phi_i(b) = b_i$ and a master feature function $\phi_m(b) = b_m$.

Finally, a master policy $\pi_m(b_m) = a^m$, a slot independent policy $\pi_i(b_i) = a^i$ and a slot dependent policy $\pi_d(b_s|\forall s \in \mathcal{S}) = a^d$ are defined, where $a^m \in \mathcal{A}_m$, $a^i \in \mathcal{A}_i$ and $a^d \in \mathcal{A}_d$. In FDQN, π_m and π_i are modelled as value-based policies. However, Policy-based models can be used to model π_m and π_i, as introduced in section 3.1. In order to generalise between slots, π_d is defined as a set of slot specific policies $\pi_s(b_s) = a^d$, one for each $s \in \mathcal{S}$. The slot specific policies have shared parameters, and the differences between slots are accounted by the abstracted dialogue state b_s. π_d runs each slot specific policy, π_s, for all $s \in \mathcal{S}$, choosing the action-slot pair that maximises the Q-value over all the slot sub-policies[2].

$$\pi_d(b_s|\forall s \in \mathcal{S}) = \underset{a^d \in \mathcal{A}_d, s \in \mathcal{S}}{\text{argmax}}\ Q^s(b_s, a^d) \quad (2)$$

Then, the summary action a is constructed by joining a^d and s (e.g. if a^d=*request()* and s=*food*, then the summary action will be *request(food)*). A pseudo-code of the Feudal Dialogue Policy algorithm is given in Appendix B.

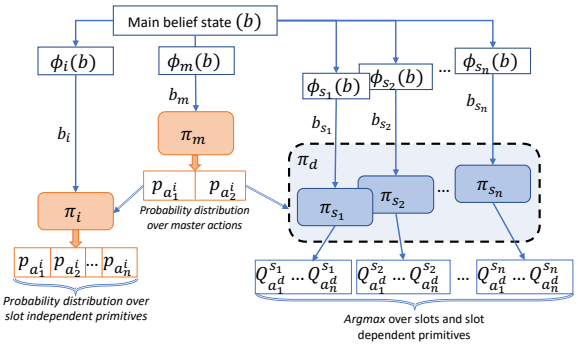

Figure 1: Feudal dialogue architecture used in this work. The sub-policies surrounded by the dashed line have shared parameters. The blue rectangles represent Value-based sub-policies while the orange ones Policy-based sub-policies.

In order to abstract the state space, FDQN uses handcrafted feature functions ϕ_i, ϕ_m and ϕ_s based on the Domain Independent Parametrisation (DIP) features introduced in Wang et al. (2015). These features include the slot independent parts of the belief state, a summarised representation of the joint belief state, and a summarised representation of the belief state of the slot s.

3 FDM with jointly learned feature extractors

In order to avoid the need to handcraft the feature functions ϕ_i, ϕ_m and ϕ_s , two methods which jointly train the feature extractors and the policy model are proposed. FDQN, however, showed to be prone to get stuck in local optima[3]. When the feature functions are jointly learned, this problem will be exacerbated due to the need to learn extra parameters. In section 3.1, two methods to avoid getting stuck in local optima are presented.

3.1 Improved training stability

FDQN showed to be prone to get stuck in local optima, overfitting to an incorrect action and continuously repeating it until the user runs out of

[2]Note that, in order to compare values from different sub-policies, π_s needs to be modelled as a Value-based policy.

[3]Depending on the initially observed dialogues, the model might get stuck in a sub-optimal policy. This is a know problem in RL (Henderson et al., 2017).

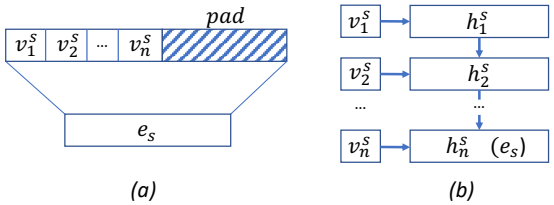

Figure 2: FFN *(a)* and RNN *(b)* jointly learned feature extractors.

Domain	Code	# constraint slots	# requests	# values
Cambridge Restaurants	CR	3	9	268
San Francisco Restaurants	SFR	6	11	636
Laptops	LAP	11	21	257

	Env. 1	Env. 2	Env. 3	Env. 4	Env. 5	Env. 6
SER	0%	0%	15%	15%	15%	30%
Masks	on	off	on	off	on	on
User	Std.	Std.	Std.	Std.	Unf.	Std.

Table 1: Sumarised description of the domains and environments used in the experiments. Refer to (Casanueva et al., 2017) for a detailed description.

patience. Appendix A shows an example of this problem. We propose two methods that combined help to reduce the overfitting, allowing the feature extractors to be learned jointly.

The belief state used in FDQN only contains information about the last system action. Therefore, if the system gets into a loop repeating the same action for every turn, the belief state cannot depict it. We propose to append the input to each sub-policy with a vector containing the frequencies of the actions taken in the current dialogue. This additional information can be used by the policy to detect these "overfitting loops" and select a different action.

Furthermore, Policy-based Actor Critic methods such as ACER (Wang et al., 2016; Weisz et al., 2018) have shown to be more stable during learning than Value-based methods. Since π_d has to compare Q-values, the slot specific policies π_s need to be Value-based. The master and slot independent policies, however, can be replaced by an Actor Critic policy, as shown in Figure 1. Section 5 shows that by doing this replacement the dialogue manager is able to learn better policies.

3.2 Jointly learned feature extractors

In order to abstract the state space into a slot-dependent fixed length representation, FDQN uses DIP feature functions (Wang et al., 2015). These features, however, need to be hand engineered by the system designer. To reduce the amount of hand-design, we propose two feature extraction models that can be learned jointly with the policy. Figure 2 shows the two proposed models. The first one *(a)*, named FFN in section 5, pads the belief state of the slot to the length of the largest slot and encodes it into a vector e_s through a feed forward neural network. The second one *(b)*, uses a recurrent neural network to encode the values of each slot into a fixed length representation e_s. Each $b_s \forall s \in \mathcal{S}$ is then constructed by concatenating the slot independent parts of the belief to the slot encoding e_s. For the feature functions ϕ_i and ϕ_m, the slot independent parts of the belief are used directly as inputs to their respective policy models. During training, the errors of the policies can be backpropagated through the feature extractors, training the models by gradient descent.

4 Experimental setup

The PyDial toolkit (Ultes et al., 2017) and the PyDial benchmarking environments (Casanueva et al., 2017)[4] have been used to implement and evaluate the models. These environments present a set of 18 tasks (Table 1) spanning differently sized domains, different Semantic Error Rates (SER), different configurations of action masks and different user model parameter sets (Standard (Std.) or Unfriendly (Unf.)).

4.1 Baselines

The feudal dialogue policy presented in (Casanueva et al., 2018) is used as a baseline, named FDQN in section 5. An implementation of FDQN using the action frequency features introduced in 3.1 is also presented, named FDQN+AF. In addition, the results of the handcrafted policy presented in (Casanueva et al., 2017) are also shown, named HDC.

4.2 Feudal ACER policy

The feudal policy proposed in section 3.1, named FACER, is implemented. This policy uses an ACER policy (Wang et al., 2016) for the slot independent and master policies, and a DQN policy (Mnih et al., 2013) for the slot specific policies. The hyperparameters of the ACER sub-policies are the same than in (Weisz et al., 2018), except for the 2 hidden layers sizes, which are reduced to 100 and 50 respectively. The hyperparameters of the DQN sub-policies are the same as FDQN.

4.3 Jointly learned feature extractors

The FDQN+AF and FACER policies are trained using the FFN and RNN feature extractors proposed in section 3.2, as well as with the DIP fea-

[4]The implementation of the models will be released

	model	FDQN+AF			FACER			FDQN	HDC
	features	DIP	FFN	RNN	DIP	FFN	RNN	DIP	-
Env. 1	CR	13.8	12.8	11.3	11.8	12.9	12.5	11,7	**14.0**
	SFR	9.4	6.0	7.3	10.9	4.5	3.8	7,1	**12.4**
	LAP	9.2	8.4	7.4	7.7	5.7	8.4	5,7	**11.7**
Env. 2	CR	13.6	11.9	12.9	13.4	13.3	13.1	13,1	**14.0**
	SFR	12.9	8.7	11.2	12.3	**13.0**	12.2	12,4	12.4
	LAP	11.8	9.6	10.8	12.1	**12.6**	**12.6**	12,0	11.7
Env. 3	CR	**13.1**	12.8	12.9	12.9	13.0	13.0	11,7	11.0
	SFR	10.3	9.8	9.9	10.3	10.1	**10.5**	9,7	9.0
	LAP	**9.8**	9.4	9.7	9.6	**9.8**	9.6	9,4	8.7
Env. 4	CR	11.9	10.8	11.3	11.9	12.0	**12.3**	11,1	11.0
	SFR	**11.2**	7.7	10.0	10.6	10.6	10.9	10,0	9.0
	LAP	9.9	-0.6	4.5	**11.2**	10.9	11.0	10,8	8.7
Env. 5	CR	11.1	10.4	11.0	11.0	**11.3**	11.2	10.4	9.3
	SFR	7.5	6.5	6.5	**7.8**	7.2	6.8	7.1	6.0
	LAP	6.8	**7.3**	6.5	6.6	6.8	6.5	6.0	5.3
Env. 6	CR	11.7	11.4	11.6	11.7	11.7	**11.8**	11.5	9.7
	SFR	**8.2**	7.5	7.4	8.1	**8.1**	7.4	7,9	6.4
	LAP	**6.7**	**6.7**	6.5	6.6	6.3	6.4	5.2	5.5

Table 2: Reward after 4000 training dialogues for FDQN+AF and FACER using DIP, FFN and RNN features, compared to FDQN and the handcrafted policy presented in the PyDial benchmarks (HDC). The best performing model is highlighted in bold while the best performing model with jointly learned features is highlighted in red.

tures used in (Casanueva et al., 2018) . For each slot $s \in \mathcal{S}$, b_s is constructed by concatenating the general and the joint belief state[5] to the encoding of the slot e_s generated by the feature extractor. The size of e_s is 25. As input for the π_m and π_i policies, the general and joint belief state is used.

5 Results

Table 2 shows the average reward[6] after 4000 training dialogues in the 18 tasks of the PyDial benchmarks. The reward for each dialogue is defined as $(suc * 20) - n$, where n is the dialogue length and $suc = 1$ if the dialogue was successful or 0 otherwise. The results are the mean over 10 different random seeds, where every seed is tested for 500 dialogues.

Comparing FDQN and FDQN-AF when using DIP features, the importance of including the action frequencies can be seen. The use of these features improves the reward in most of the tasks between 0.5 and 2 points. When training the policies with the joint feature extractors, the action frequencies were found to be a key feature in order to avoid the policies to get stuck in local optima.

FACER shows the best performance with the jointly learned feature extractors, outperforming any other policy (including the ones using DIP features) in 8 out of 18 tasks, and obtaining a very similar performance in the rest. This shows the improved training stability given by the Policy-based models. In task 1, however, (where FDQN already showed overfitting problems) the FDQN+AF is able to learn better feature extractors than FACER, but the performance is still worse than HDC.

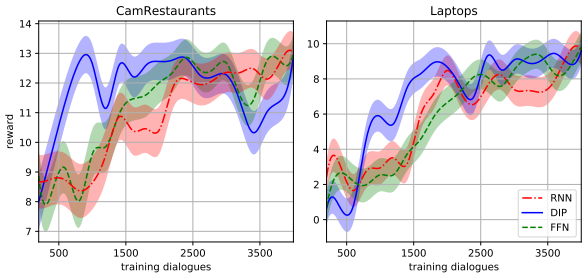

Figure 3: Learning curves for FACER in env. 3 in Cambridge Restaurants and Laptops domains, using DIP, FFN and RNN features. The shaded area represents the standard deviation.

Figure 3 shows the learning curves for FACER in two domains of Env. 3 using the two learned feature extractors (FFN and RNN) compared to the DIP features. It can be observed that the learned features take longer to converge, but the difference is smaller than it could be expected, especially in a large domain such as Laptops.

6 Conclusions and future work

This paper has shown that the feature functions needed to abstract the dialogue state space in feudal dialogue management can be jointly learned with the policy, thus reducing the need of handcrafting them. In order to make it possible to learn the features jointly, two methods to increase the robustness of the model against overfitting were introduced: extending the input features with action frequencies and substituting the master and domain independent policies by ACER policies. In combination, these modifications showed to improve the results in most of the PyDial benchmarking tasks by an average of 1 point in reward, while reducing the handcrafting effort.

However, as the original FDQN architecture needs to model the slot specific policies as Value-based models, ACER policies could only be used for the master and slot independent policies. Future work will investigate new FDM architectures which allow the use of Policy-based models as slot specific policies, while maintaining the parameter sharing mechanism between slots.

[5]The joint belief state is sorted and truncated to size 20.

[6]Because of space issues, the success rate is not included. However, the success rate is very correlated with the results presented in (Casanueva et al., 2017) and (Casanueva et al., 2018).

Acknowledgments

This research was funded by the EPSRC grant EP/M018946/1 Open Domain Statistical Spoken Dialogue Systems

References

Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Stefan Ultes, Lina Rojas-Barahona, Steve Young, and Milica Gašić. 2017. A benchmarking environment for reinforcement learning based task oriented dialogue management. *Deep Reinforcement Learning Symposium, 31st Conference on Neural Information Processing Systems (NIPS 2017)*.

Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Stefan Ultes, Lina Rojas-Barahona, Bo-Hsiang Tseng, and Milica Gašić. 2018. Feudal reinforcement learning for dialogue management in large domains. *arXiv preprint arXiv:1803.03232*.

Peter Dayan and Geoffrey E Hinton. 1993. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278.

Milica Gašić, Catherine Breslin, Matthew Henderson, Dongho Kim, Martin Szummer, Blaise Thomson, Pirros Tsiakoulis, and Steve Young. 2013. Pomdp-based dialogue manager adaptation to extended domains. In *Proceedings of the SIGDIAL Conference*.

James Henderson, Oliver Lemon, and Kallirroi Georgila. 2008. Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets. *Computational Linguistics*, 34(4):487–511.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2017. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. 1998. Using markov decision process for learning dialogue strategies. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 1, pages 201–204. IEEE.

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

Olivier Pietquin, Matthieu Geist, Senthilkumar Chandramohan, et al. 2011. Sample efficient online learning of optimal dialogue policies with kalman temporal differences. IJCAI Proceedings-International Joint Conference on Artificial Intelligence.

Richard S. Sutton and Andrew G. Barto. 1999. *Reinforcement Learning: An Introduction*. MIT Press.

Stefan Ultes, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gašić, and Steve J. Young. 2017. Pydial: A multi-domain statistical dialogue system toolkit. In *ACL Demo*. Association of Computational Linguistics.

Zhuoran Wang, Tsung-Hsien Wen, Pei-Hao Su, and Yannis Stylianou. 2015. Learning domain-independent dialogue policies via ontology parameterisation. In *SIGDIAL Conference*, pages 412–416.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. 2016. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

Gellért Weisz, Paweł Budzianowski, Pei-Hao Su, and Milica Gašić. 2018. Sample efficient deep reinforcement learning for dialogue systems with large action spaces. *arXiv preprint arXiv:1802.03753*.

Jason D. Williams and Steve Young. 2007. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(2):393–422.

Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. 2013. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179.

A Dialogues getting stuck in local optima

In this section we present an example of a policy model getting stuck in a sub-optimal policy. The two following dialogues represent a dialogue observed in the initial training steps of the policy and a dialogue observed once the policy has overfitted.

Initial dialogue:
Goal: *food*=british, *area*=centre

1: **usr:** Inform(food=british)
2: **sys:** Confirm(food=british)
3: **usr:** Affirm()|Inform(area=centre)
4: **sys:** Inform(name=The Eagle)
5: **usr:** Thankyou()|Bye()

Overfitted dialogue:
Goal: *food*=british, *area*=centre

1: **usr:** Inform(food=british)
2: **sys:** Confirm(food=british)
3: **usr:** Affirm()
4: **sys:** Confirm(food=british)
5: **usr:** Affirm()
6: **sys:** Confirm(food=british)
7: **usr:** Affirm()
8: **sys:** Confirm(food=british)
9: **usr:** Bye()

In the initial dialogue, the policy interacts with a collaborative user[7], which in line 3, provides more information than the requested by the policy. The dialogue ends up successfully and, therefore, the policy learns that by confirming the slot *food* in that dialogue state it will get enough information to end the dialogue successfully. In the second dialogue, however, the system interacts with a less collaborative user. Therefore, when confirming the slot *food* in line 3, it doesn't get the extra information obtained in the previous dialogue. The policy keeps insisting with this action, until the user runs out of patience and ends up the dialogue. Even with ϵ-*greedy* exploration, as a fraction of the sampled users will be collaborative enough to make this policy successful, the policy can get stuck in this local optima and never learn a better policy - i.e. requesting the value of the slot *area*. Other examples of overfitting include policies informing entities at random from the first turn (since some users will correct the policy by informing the correct values) or policies that don't learn to inform about the requested slots (since the sampled user goal sometimes doesn't include requesting any extra information, just the entity name).

[7]The user parameters are sampled at the beginning of each dialogue.

B Feudal Dialogue Policy algorithm

Algorithm 1 Feudal Dialogue Policy

for each dialogue turn **do**
  observe b
  $b_m = \phi_m(b)$
  $a^m = \pi_m(b_m)$
  if $a^m == a^m_i$ **then** ▷ drop to π_i
    $b_i = \phi_i(b)$
    $a = \pi_i(b_i)$
  else $a^m == a^m_d$ **then** ▷ drop to π_d
    $b_s = \phi_s(b)\, \forall s \in \mathcal{S}$
    $slot, act = \underset{s \in \mathcal{S}, a^d \in \mathcal{A}_d}{\text{argmax}}\ Q^s(b_s, a^d)$
    $a = join(slot, act)$
  end if
  execute a
end for