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 FDQN1, 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 𝑀, a continuous multivariate belief state space 𝐵 and a reward function 𝑅(𝑏𝑡, 𝑎𝑡). At a given time 𝑡, the agent observes the belief state 𝑏𝑡 ∈ 𝐵, executes an action 𝑎𝑡 ∈ 𝑀 and receives a reward 𝑟𝑡 ∈ ℝ drawn from 𝑅(𝑏𝑡, 𝑎𝑡). The action taken, 𝑎, is decided by the policy, defined as the function 𝜋(𝑏) = 𝑎. The objective of RL is to find the optimal policy 𝜋∗ that maximizes

1In the rest of the paper we will refer to the FDM model presented in Casanueva et al. (2018) as FDQN.
the expected return $R$ in each belief state, where $R = \sum_{t=1}^{T} \gamma^{(t-1)} r_t$, $\gamma$ 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 $\theta$ 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 $\pi$ until the end of the dialogue at time step $T$:

$$Q^\pi(b, a) = \mathbb{E}\{R\mid b_t = b, a_t = a\}$$  (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 $A_i$ (e.g. hello(), inform()); and slot dependent actions $A_d$ (e.g. request(), confirm()). In addition, a set of master actions $A_m = (a_{m_i}^i, a_{m_d}^d)$ is defined, where $a_{m_i}^i$ corresponds to taking an action from $A_i$ and $a_{m_d}^d$ to taking an action from $A_d$. 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 $s \in S$, 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 $\pi_d(b_d | \forall s \in S) = a_d$ are defined, where $a_m \in A_m$, $a_i \in A_i$ and $a_d \in A_d$. In FDNQ, $\pi_m$ and $\pi_i$ are modelled as value-based policies. However, Policy-based models can be used to model $\pi_m$ and $\pi_i$, as introduced in section 3.1. In order to generalise between slots, $\pi_d$ is defined as a set of slot specific policies $\pi_s(b_s) = a_d$, one for each $s \in S$. The slot specific policies have shared parameters, and the differences between slots are accounted by the abstracted dialogue state $b_s$. $\pi_d$ runs each slot specific policy, $\pi_s$, for all $s \in S$, choosing the action-slot pair that maximises the Q-value over all the slot sub-policies$^2$.

$$\pi_d(b_s | \forall s \in S) = \arg\max_{a_d \in A_d, s \in S} Q^\pi(b_s, a_d)$$  (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.

In order to abstract the state space, FDQN uses handcrafted feature functions $\phi_i$, $\phi_m$ and $\phi_s$ 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 $\phi_i$, $\phi_m$ and $\phi_s$, 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, $\pi_s$ 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 known problem in RL (Henderson et al., 2017).
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 πd has to compare Q-values, the slot specific policies πs 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

Figure 2: FFN (a) and RNN (b) 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 es 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 es. Each bs∀s ∈ S is then constructed by concatenating the slot independent parts of the belief to the slot encoding es. For the feature functions ϕt and ϕm,
The size of the general and the joint belief state is highlighted in red.

The best performing model is highlighted in bold while the best performing model with (HDC). The best performing model is highlighted for FDQN and the hand-crafted policy presented in the PyDial benchmarks (Casanueva et al., 2017) and (Casanueva et al., 2018).

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.

5 Results

Table 2 shows the average reward 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 policy 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.

Table 2: Reward after 4000 training dialogues for FDQN+AF and FACER using DIP, FFN and RNN features, compared to FDQN and the hand-crafted 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.

<table>
<thead>
<tr>
<th>model features</th>
<th>FDQN+AF</th>
<th>FACER</th>
<th>DIP</th>
<th>FFN</th>
<th>RNN</th>
<th>DIP</th>
<th>FFN</th>
<th>RNN</th>
<th>DIP</th>
<th>FFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>13.8</td>
<td>12.8</td>
<td>11.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFR</td>
<td>9.4</td>
<td>6.0</td>
<td>7.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAP</td>
<td>9.2</td>
<td>8.4</td>
<td>7.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>13.6</td>
<td>11.9</td>
<td>12.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFR</td>
<td>12.9</td>
<td>8.7</td>
<td>11.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAP</td>
<td>11.8</td>
<td>9.6</td>
<td>10.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>13.1</td>
<td>12.8</td>
<td>12.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFR</td>
<td>10.3</td>
<td>9.8</td>
<td>9.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAP</td>
<td>9.8</td>
<td>9.4</td>
<td>9.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>11.9</td>
<td>10.8</td>
<td>11.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFR</td>
<td>11.2</td>
<td>7.7</td>
<td>10.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAP</td>
<td>9.9</td>
<td>0.6</td>
<td>4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>11.1</td>
<td>10.4</td>
<td>11.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFR</td>
<td>7.5</td>
<td>6.5</td>
<td>6.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAP</td>
<td>6.8</td>
<td>7.3</td>
<td>6.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>11.7</td>
<td>11.4</td>
<td>11.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFR</td>
<td>8.2</td>
<td>7.5</td>
<td>7.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAP</td>
<td>6.7</td>
<td>6.7</td>
<td>6.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5 Conclusions and future work

This paper has shown that the feature functions needed to abstract the dialogue state space in featural 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.
Acknowledgments
This research was funded by the EPSRC grant EP/M018946/1 Open Domain Statistical Spoken Dialogue Systems

References


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, 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 $\epsilon$-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).

B  Feudal Dialogue Policy algorithm

Algorithm 1 Feudal Dialogue Policy

1: for each dialogue turn do
2: observe $b$
3: $b_m = \phi_m(b)$
4: $a^m = \pi_m(b_m)$
5: if $a^m = a^m_i$ then $\triangleright$ drop to $\pi_i$
6: $b_i = \phi_i(b)$
7: $a = \pi_i(b_i)$
8: else $a^m = a^m_d$ then $\triangleright$ drop to $\pi_d$
9: $b_s = \phi_s(b) \forall s \in S$
10: $slot, act = \arg\max_{s \in S, a \in A_d} Q^s(b_s, a^d)$
11: $a = join(slot, act)$
12: end if
13: execute $a$
14: end for

\footnote{The user parameters are sampled at the beginning of each dialogue.}