Hybrid Code Networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning

Jason D. Williams
Microsoft Research
jason.williams@microsoft.com

Kavosh Asadi
Brown University
kavosh@brown.edu

Geoffrey Zweig[*]
Microsoft Research
g2zweig@gmail.com

Abstract

End-to-end learning of recurrent neural networks (RNNs) is an attractive solution for dialog systems; however, current techniques are data-intensive and require thousands of dialogs to learn simple behaviors. We introduce Hybrid Code Networks (HCNs), which combine an RNN with *domain-specific knowledge encoded as software* and *system action templates*. Compared to existing end-to-end approaches, HCNs considerably reduce the amount of training data required, while retaining the key benefit of inferring a latent representation of dialog state. In addition, HCNs can be optimized with supervised learning, reinforcement learning, or a mixture of both. HCNs attain state-of-the-art performance on the bAbI dialog dataset (Bordes and Weston, 2016), and outperform two commercially deployed customer-facing dialog systems.

1 Introduction

Task-oriented dialog systems help a user to accomplish some goal using natural language, such as making a restaurant reservation, getting technical support, or placing a phonecall. Historically, these dialog systems have been built as a pipeline, with modules for language understanding, state tracking, action selection, and language generation. However, dependencies between modules introduce considerable complexity – for example, it is often unclear how to define the dialog state and what history to maintain, yet action selection relies exclusively on the state for input. Moreover, training each module requires specialized labels.

Recently, end-to-end approaches have trained recurrent neural networks (RNNs) directly on text transcripts of dialogs. A key benefit is that the RNN infers a latent representation of state, obviating the need for state labels. However, end-to-end methods lack a general mechanism for injecting domain knowledge and constraints. For example, simple operations like sorting a list of database results or updating a dictionary of entities can expressed in a few lines of software, yet may take thousands of dialogs to learn. Moreover, in some practical settings, programmed constraints are essential – for example, a banking dialog system would require that a user is logged in before they can retrieve account information.

This paper presents a model for end-to-end learning, called *Hybrid Code Networks* (HCNs) which addresses these problems. In addition to learning an RNN, HCNs also allow a developer to express domain knowledge via software and action templates. Experiments show that, compared to existing recurrent end-to-end techniques, HCNs achieve the same performance with considerably less training data, while retaining the key benefit of end-to-end trainability. Moreover, the neural network can be trained with supervised learning or reinforcement learning, by changing the gradient update applied.

This paper is organized as follows. Section 2 describes the model, and Section 3 compares the model to related work. Section 4 applies HCNs to the bAbI dialog dataset (Bordes and Weston, 2016). Section 5 then applies the method to real customer support domains at our company. Section 6 illustrates how HCNs can be optimized with reinforcement learning, and Section 7 concludes.

---

[*] Currently at JPMorgan Chase

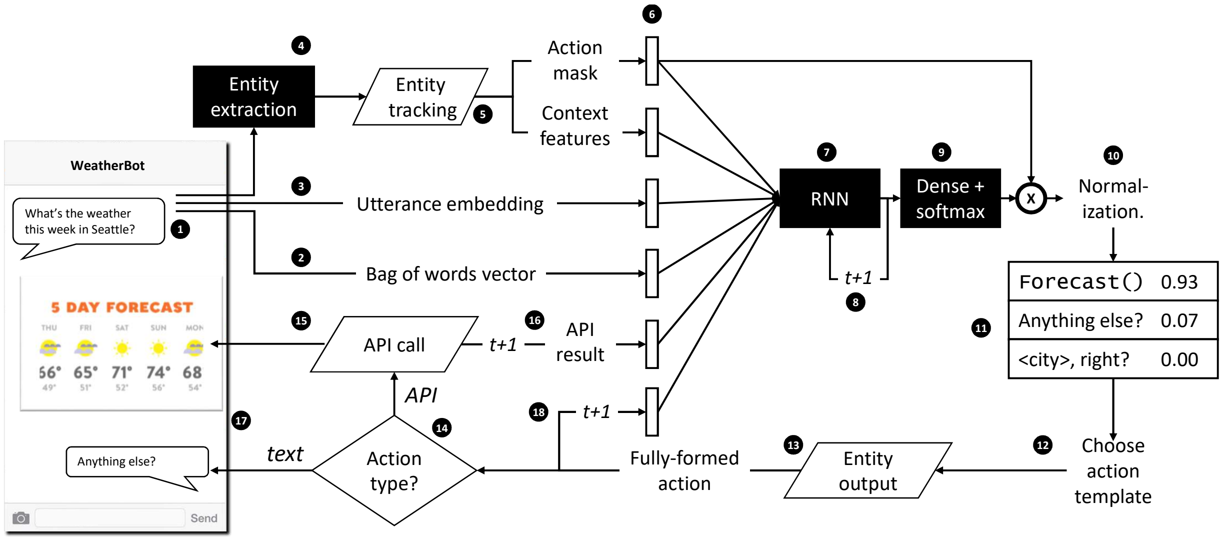


Figure 1: Operational loop. Trapezoids refer to programmatic code provided by the software developer, and shaded boxes are trainable components. Vertical bars under "6" represent concatenated vectors which form the input to the RNN.

2 Model description

At a high level, the four components of a Hybrid Code Network are a recurrent neural network; domain-specific software; domain-specific action templates; and a conventional entity extraction module for identifying entity mentions in text. Both the RNN and the developer code maintain state. Each action template can be a textual communicative action or an API call. The HCN model is summarized in Figure 1.

The cycle begins when the user provides an utterance, as text (step 1). The utterance is featurized in several ways. First, a bag of words vector is formed (step 2). Second, an utterance embedding is formed, using a pre-built utterance embedding model (step 3). Third, an entity extraction module identifies entity mentions (step 4) – for example, identifying "Jennifer Jones" as a `<name>` entity. The text and entity mentions are then passed to "Entity tracking" code provided by the developer (step 5), which grounds and maintains entities – for example, mapping the text "Jennifer Jones" to a specific row in a database. This code can optionally return an "action mask", indicating actions which are permitted at the current timestep, as a bit vector. For example, if a target phone number has not yet been identified, the API action to place a phone call may be masked. It can also optionally return "context features" which are features the developer thinks will be useful for distinguishing among actions, such as which entities are currently present and which are absent.

The feature components from steps 1-5 are concatenated to form a feature vector (step 6). This vector is passed to an RNN, such as a long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) or gated recurrent unit (GRU) (Chung et al., 2014). The RNN computes a hidden state (vector), which is retained for the next timestep (step 8), and passed to a dense layer with a softmax activation, with output dimension equal to the number of distinct system action templates (step 9).[1] Thus the output of step 9 is a distribution over action templates. Next, the action mask is applied as an element-wise multiplication, and the result is normalized back to a probability distribution (step 10) – this forces non-permitted actions to take on probability zero. From the resulting distribution (step 11), an action is selected (step 12). When RL is active, exploration is required, so in this case an action is *sampled* from the distribution; when RL is not active, the best action should be chosen, and so the action with the *highest probability* is always selected.

The selected action is next passed to "Entity output" developer code that can substitute in entities (step 13) and produce a fully-formed action – for example, mapping the template "`<city>`,

---

[1] Implementation details for the RNN such as size, loss, etc. are given with each experiment in Sections 4-6.

right?" to "Seattle, right?". In step 14, control branches depending on the type of the action: if it is an API action, the corresponding API call in the developer code is invoked (step 15) – for example, to render rich content to the user. APIs can act as sensors and return features relevant to the dialog, so these can be added to the feature vector in the next timestep (step 16). If the action is text, it is rendered to the user (step 17), and cycle then repeats. The action taken is provided as a feature to the RNN in the next timestep (step 18).

3 Related work

Broadly there are two lines of work applying machine learning to dialog control. The first decomposes a dialog system into a pipeline, typically including language understanding, dialog state tracking, action selection policy, and language generation (Levin et al., 2000; Singh et al., 2002; Williams and Young, 2007; Williams, 2008; Hori et al., 2009; Lee et al., 2009; Griol et al., 2008; Young et al., 2013; Li et al., 2014). Specifically related to HCNs, past work has implemented the policy as feed-forward neural networks (Wen et al., 2016), trained with supervised learning followed by reinforcement learning (Su et al., 2016). In these works, the policy has not been *recurrent* – i.e., the policy depends on the state tracker to summarize observable dialog history into state features, which requires design and specialized labeling. By contrast, HCNs use an RNN which automatically infers a representation of state. For learning efficiency, HCNs use an external lightweight process for tracking entity values, but the policy is not strictly dependent on it: as an illustration, in Section 5 below, we demonstrate an HCN-based dialog system which has no external state tracker. If there is context which is not apparent in the text in the dialog, such as database status, this can be encoded as a context feature to the RNN.

The second, more recent line of work applies recurrent neural networks (RNNs) to learn "end-to-end" models, which map from an observable dialog history directly to a sequence of output words (Sordoni et al., 2015; Shang et al., 2015; Vinyals and Le, 2015; Yao et al., 2015; Serban et al., 2016; Li et al., 2016a,c; Luan et al., 2016; Xu et al., 2016; Li et al., 2016b; Mei et al., 2016; Lowe et al., 2017; Serban et al., 2017). These systems can be applied to task-oriented domains by adding special "API call" actions, enumerating database output as a sequence of tokens (Bordes and Weston, 2016), then learning an RNN using Memory Networks (Sukhbaatar et al., 2015), gated memory networks (Liu and Perez, 2016), query reduction networks (Seo et al., 2016), and copy-augmented networks (Eric and Manning, 2017). In each of these architectures, the RNN learns to manipulate entity values, for example by saving them in a memory. Output is produced by generating a sequence of tokens (or ranking all possible surface forms), which can also draw from this memory. HCNs also use an RNN to accumulate dialog state and choose actions. However, HCNs differ in that they use developer-provided action templates, which can contain entity references, such as "`<city>`, right?". This design reduce learning complexity, and also enable the software to limit which actions are available via an action mask, at the expense of developer effort. To further reduce learning complexity in a practical system, entities are tracked separately, outside the the RNN, which also allows them to be substituted into action templates. Also, past end-to-end recurrent models have been trained using supervised learning, whereas we show how HCNs can also be trained with reinforcement learning.

4 Supervised learning evaluation I

In this section we compare HCNs to existing approaches on the public "bAbI dialog" dataset (Bordes and Weston, 2016). This dataset includes two end-to-end dialog learning tasks, in the restaurant domain, called task5 and task6.[2] Task5 consists of synthetic, simulated dialog data, with highly regular user behavior and constrained vocabulary. Dialogs include a database access action which retrieves relevant restaurants from a database, with results included in the dialog transcript. We test on the "OOV" variant of Task5, which includes entity values not observed in the training set. Task6 draws on human-computer dialog data from the second dialog state tracking challenge (DSTC2), where usability subjects (crowd-workers) interacted with several variants of a spoken dialog system (Henderson et al., 2014). Since the database from DSTC2 was not provided, database calls have been inferred from the data and inserted into the dialog transcript. Example dialogs are provided in the Appendix Sections A.2 and A.3.

To apply HCNs, we wrote simple domain-

---

[2] Tasks 1-4 are sub-tasks of Task5.

specific software, as follows. First, for entity extraction (step 3 in Figure 1), we used a simple string match, with a pre-defined list of entity names – i.e., the list of restaurants available in the database. Second, in the context update (step 4), we created simple rules for tracking entities, where entities recognized in the input overwrite the existing entries. Third, system actions were templatized: for example, system actions of the form "prezzo is a nice restaurant in the west of town in the moderate price range" all map to the template "`<name>` is a nice restaurant in the `<location>` of town in the `<price>` price range". This results in 16 templates for Task5 and 58 for Task6.[3] Fourth, when database results are received into the entity state, they sorted by rating. Finally, an action mask was created which encoded common-sense dependencies: for example, only allow an API call if pre-conditions are met; only offer a restaurant if database results have already been received; do not ask for an entity if it is already known; etc.

For Task6, we noticed that the system can say that no restaurants match the current query *without* consulting the database (for an example dialog, see Section A.3 in the Appendix). In a practical system this information would be retrieved from the database and not encoded in the RNN. So, we mined the training data and built a table of search queries known to yield no results. We also added context features that indicated the state of the database – for example, whether there were any restaurants matching the current query. The complete set of context features is given in Appendix Section A.4. Altogether this code consisted of about 250 lines of Python.

We then trained an HCN on the training set, employing the domain-specific software described above. We selected an LSTM for the recurrent layer (Hochreiter and Schmidhuber, 1997), with the AdaDelta optimizer (Zeiler, 2012). We used the development set to tune the number of hidden units (128), and the number of epochs (12). Utterance embeddings were formed by averaging word embeddings, using a publicly available 300-dimensional word embedding model trained using word2vec on web data (Mikolov et al., 2013).[4] The word embeddings were static and not updated during LSTM training. In training, each dialog formed one minibatch, and updates were done on full rollouts (i.e., non-truncated back propagation through time). The training loss was categorical cross-entropy. Further low-level implementation details are in the Appendix Section A.1.

We ran experiments with four variants of our model: with and without the utterance embeddings, and with and without the action mask (Figure 1, steps 2 and 5 respectively).

Following past work, we report average turn accuracy – i.e., for each turn in each dialog, present the (true) history of user and system actions to the network and obtain the network's prediction as a string of characters. The turn is correct if the string matches the reference exactly, and incorrect if not. We also report dialog accuracy, which indicates if all turns in a dialog are correct.

We compare to four past end-to-end approaches (Bordes and Weston, 2016; Liu and Perez, 2016; Eric and Manning, 2017; Seo et al., 2016). We emphasize that past approaches have applied purely sequence-to-sequence models, or (as a baseline) purely programmed rules (Bordes and Weston, 2016). By contrast, Hybrid Code Networks are a hybrid of hand-coded rules and learned models.

Results are shown in Table 1. Since Task5 is synthetic data generated using rules, it is possible to obtain perfect accuracy using rules (line 1). The addition of domain knowledge greatly simplifies the learning task and enables HCNs to also attain perfect accuracy. On Task6, rules alone fare poorly, whereas HCNs outperform past learned models.

We next examined learning curves, training with increasing numbers of dialogs. To guard against bias in the ordering of the training set, we averaged over 5 runs, randomly permuting the order of the training dialogs in each run. Results are in Figure 2. In Task5, the action mask and utterance embeddings substantially reduce the number of training dialogs required (note the horizontal axis scale is logarithmic). For Task6, the benefits of the utterance embeddings are less clear. An error analysis showed that there are several systematic differences between the training and testing sets. Indeed, DSTC2 intentionally used different dialog policies for the training and test sets, whereas our goal is to mimic the policy in the training set.

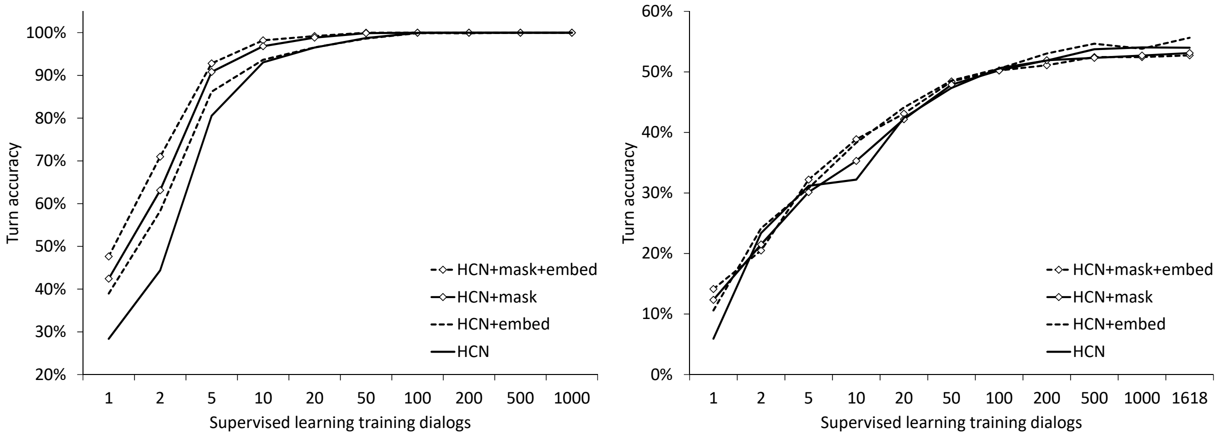
---

[3] A handful of actions in Task6 seemed spurious; for these, we replaced them with a special "UNK" action in the training set, and masked this action at test time.

[4] Google News 100B model from `https://github.com/3Top/word2vec-api`

	Task5-OOV		**Task6**	
Model	Turn Acc.	Dialog Acc.	Turn Acc.	Dialog Acc.
Rules	**100%**	**100%**	33.3%	0.0%
Bordes and Weston (2016)	77.7%	0.0%	41.1%	0.0%
Liu and Perez (2016)	79.4%	0.0%	48.7%	1.4%
Eric and Manning (2017)	—	—	48.0%	1.5%
Seo et al. (2016)	96.0%	—	51.1%	—
HCN	**100%**	**100%**	54.0%	1.2%
HCN+embed	**100%**	**100%**	**55.6%**	1.3%
HCN+mask	**100%**	**100%**	53.1%	**1.9%**
HCN+embed+mask	**100%**	**100%**	52.7%	1.5%

Table 1: Results on bAbI dialog Task5-OOV and Task6 (Bordes and Weston, 2016). Results for "Rules" taken from Bordes and Weston (2016). Note that, unlike cited past work, HCNs make use of domain-specific procedural knowledge.



(a) bAbI dialog Task5-OOV.

(b) bAbI dialog Task6.

Figure 2: Training dialog count vs. turn accuracy for bAbI dialog Task5-OOV and Task6. "embed" indicates whether utterance embeddings were included; "mask" indicates whether the action masking code was active.

Nonetheless, these tasks are the best public benchmark we are aware of, and HCNs exceed performance of existing sequence-to-sequence models. In addition, they match performance of past models using an order of magnitude less data (200 vs. 1618 dialogs), which is crucial in practical settings where collecting realistic dialogs for a new domain can be expensive.

5 Supervised learning evaluation II

We now turn to comparing with purely hand-crafted approaches. To do this, we obtained logs from our company's text-based customer support dialog system, which uses a sophisticated rule-based dialog manager. Data from this system is attractive for evaluation because it is used by real customers – not usability subjects – and because its rule-based dialog manager was developed by customer support professionals at our company, and not the authors. This data is not publicly available, but we are unaware of suitable human-computer dialog data in the public domain which uses rules.

Customers start using the dialog system by entering a brief description of their problem, such as "I need to update my operating system". They are then routed to one of several hundred domains, where each domain attempts to resolve a particular problem. In this study, we collected human-computer transcripts for the high-traffic domains "reset password" and "cannot access account".

We labeled the dialog data as follows. First, we enumerated unique system actions observed in the data. Then, for each dialog, starting from the beginning, we examined each system action, and determined whether it was "correct". Here, correct means that it was the most appropriate action among the set of existing system actions, given the history of that dialog. If multiple actions were arguably appropriate, we broke ties in favor of the existing rule-based dialog manager. Example dialogs are provided in the Appendix Sections A.5 and A.6.

If a system action was labeled as correct, we left it as-is and continued to the next system action. If the system action was not correct, we replaced it with the correct system action, and discarded the rest of the dialog, since we do not know how the user would have replied to this new system action. The resulting dataset contained a mixture of complete and partial dialogs, containing only correct system actions. We partitioned this set into training and test dialogs. Basic statistics of the data are shown in Table 2.

In this domain, no entities were relevant to the control flow, and there was no obvious mask logic since any question could follow any question. Therefore, we wrote no domain-specific software for this instance of the HCN, and relied purely on the recurrent neural network to drive the conversation. The architecture and training of the RNN was the same as in Section 4, except that here we did not have enough data for a validation set, so we instead trained until we either achieved 100% accuracy on the training set or reached 200 epochs.

To evaluate, we observe that conventional measures like average dialog or turn accuracy penalize the rule-based system unfairly, since when the rule-based system makes an error in turn t, the HCN is only evaluated on the subsequence ending at turn t. We therefore use a comparative measure that examines which method produces longer continuous sequences of correct system actions, starting from the beginning of the dialog. Specifically, we report $\Delta P = \frac{C(\text{HCN-win}) - C(\text{rule-win})}{C(\text{all})}$, where $C(\text{HCN-win})$ is the number of test dialogs where the rule-based approach output a wrong action before the HCN; $C(\text{rule-win})$ is the number of test dialogs where the HCN output a wrong action before the rule-based approach; and $C(\text{all})$ is the number of dialogs in the test set. When $\Delta P > 0$, HCNs more often produce continuous sequences of correct actions starting from the beginning of the dialog. We run all experiments 5 times, each time shuffling the order of the training set. Results are in Figure 3. HCNs exceed performance of the existing rule-based system after about 30 dialogs.

	Forgot password	Account Access
Av. sys. turns/dialog	2.2	2.2
Max. sys. turns/dialog	5	9
Av. words/user turn	7.7	5.4
Unique sys. actions	7	16
Train dialogs	422	56
Test dialogs	148	60
Test acc. (rules)	64.9%	42.1%

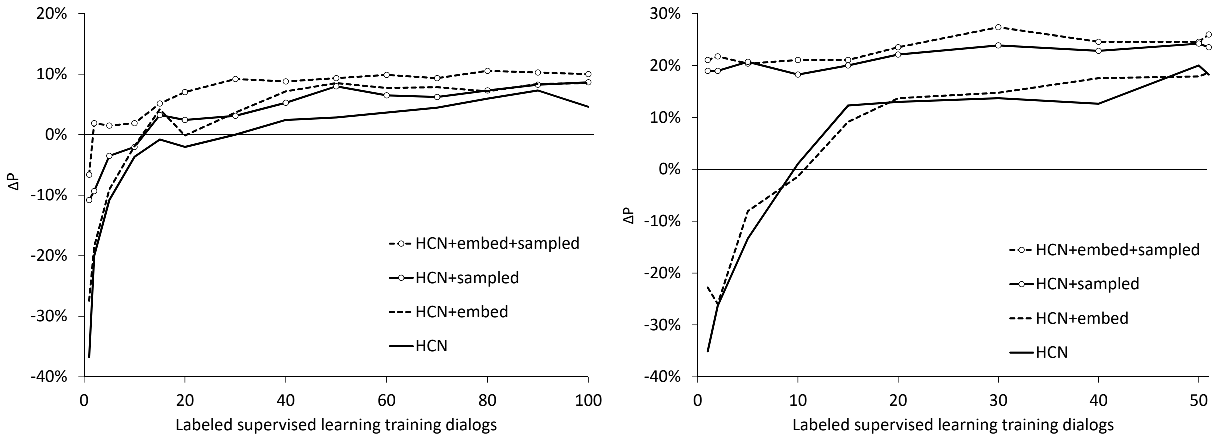
Table 2: Basic statistics of labeled customer support dialogs. Test accuracy refers to whole-dialog accuracy of the existing rule-based system.

In these domains, we have a further source of knowledge: the rule-based dialog managers themselves can be used to generate example "sunny-day" dialogs, where the user provides purely expected inputs. From each rule-based controller, synthetic dialogs were sampled to cover each expected user response at least once, and added to the set of labeled real dialogs. This resulted in 75 dialogs for the "Forgot password" domain, and 325 for the "Can't access account" domain. Training was repeated as described above. Results are also included in Figure 3, with the suffix "sampled". In the "Can't access account" domain, the sampled dialogs yield a large improvement, probably because the flow chart for this domain is large, so the sampled dialogs increase coverage. The gain in the "forgot password" domain is present but smaller.

In summary, HCNs can out-perform production-grade rule-based systems with a reasonable number of labeled dialogs, and adding synthetic "sunny-day" dialogs improves performance further. Moreover, unlike existing pipelined approaches to dialog management that rely on an explicit state tracker, this HCN used no explicit state tracker, highlighting an advantage of the model.

6 Reinforcement learning illustration

In the previous sections, supervised learning (SL) was applied to train the LSTM to mimic dialogs provided by the system developer. Once a system operates at scale, interacting with a large number



(a) "Forgot password" domain.

(b) "Can't access account" domain.

Figure 3: Training dialogs vs. ΔP, where ΔP is the fraction of test dialogs where HCNs produced longer initial correct sequences of system actions than the rules, minus the fraction where rules produced longer initial correct sequences than the HCNs. "embed" indicates whether utterance embeddings were included; "sampled" indicates whether dialogs sampled from the rule-based controller were included in the training set.

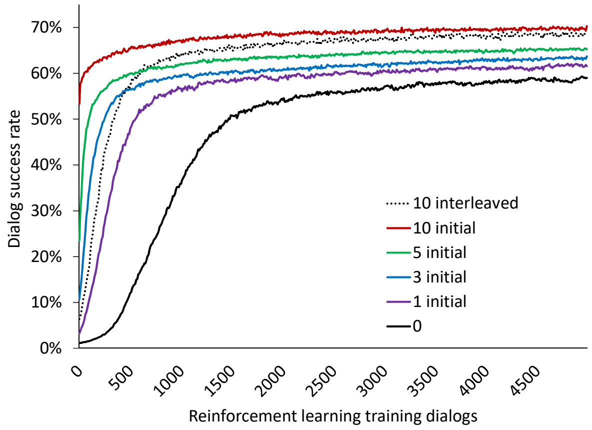


Figure 4: Dialog success rate vs. reinforcement learning training dialogs. Curve marked "0" begins with a randomly initialized LSTM. Curves marked "N initial" are pre-trained with N labeled dialogs. Curve marked "10, interleaved" adds one SL training dialog before RL dialog 0, 100, 200, ... 900.

of users, it is desirable for the system to continue to learn *autonomously* using reinforcement learning (RL). With RL, each turn receives a measurement of goodness called a *reward*; the agent explores different sequences of actions in different situations, and makes adjustments so as to maximize the expected discounted sum of rewards, which is called the *return*, denoted G.

For optimization, we selected a *policy gradient* approach (Williams, 1992), which has been successfully applied to dialog systems (Jurčíček et al., 2011), robotics (Kohl and Stone, 2004), and the board game Go (Silver et al., 2016). In policy gradient-based RL, a model π is parameterized by $\mathbf{w}$ and outputs a distribution from which actions are sampled at each timestep. At the end of a dialog, the return G for that dialog is computed, and the gradients of the probabilities of the actions taken with respect to the model weights are computed. The weights are then adjusted by taking a gradient step proportional to the return:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\Big(\sum_t \nabla_{\mathbf{w}} \log \pi(a_t|\mathbf{h_t};\mathbf{w})\Big)(G-b) \quad (1)$$

where α is a learning rate; a_t is the action taken at timestep t; $\mathbf{h_t}$ is the dialog history at time t; G is the return of the dialog; $\nabla_{\mathbf{x}} F$ denotes the Jacobian of F with respect to $\mathbf{x}$; b is a baseline described below; and $\pi(a|\mathbf{h};\mathbf{w})$ is the LSTM – i.e., a stochastic policy which outputs a distribution over a given a dialog history $\mathbf{h}$, parameterized by weights $\mathbf{w}$. The baseline b is an estimate of the average return of the current policy, estimated on the last 100 dialogs using weighted importance sampling.[5] Intuitively, "better" dialogs receive a positive gradient

---

[5] The choice of baseline does not affect the long-term convergence of the algorithm (i.e., the bias), but can dramatically affect the speed of convergence (i.e., the variance) (Williams, 1992).

step, making the actions selected more likely; and "worse" dialogs receive a negative gradient step, making the actions selected less likely.

SL and RL correspond to different methods of updating weights, so both can be applied to the same network. However, there is no guarantee that the optimal RL policy will agree with the SL training set; therefore, after each RL gradient step, we check whether the updated policy reconstructs the training set. If not, we re-run SL gradient steps on the training set until the model reproduces the training set. Note that this approach allows new training dialogs to be added at any time during RL optimization.

We illustrate RL optimization on a simulated dialog task in the name dialing domain. In this system, a contact's name may have synonyms ("Michael" may also be called "Mike"), and a contact may have more than one phone number, such as "work" or "mobile", which may in turn have synonyms like "cell" for "mobile". This domain has a database of names and phone numbers, 5 entities, and 14 actions, including 2 API call actions. Simple entity logic was coded, which retains the most recent copy of recognized entities. A simple action mask suppresses impossible actions, such as placing a phonecall before a phone number has been retrieved from the database. Example dialogs are provided in Appendix Section A.7.

To perform optimization, we created a simulated user. At the start of a dialog, the simulated user randomly selected a name and phone type, including names and phone types not covered by the dialog system. When speaking, the simulated user can use the canonical name or a nickname; usually answers questions but can ignore the system; can provide additional information not requested; and can give up. The simulated user was parameterized by around 10 probabilities, set by hand.

We defined the reward as being 1 for successfully completing the task, and 0 otherwise. A discount of 0.95 was used to incentivize the system to complete dialogs faster rather than slower, yielding return 0 for failed dialogs, and $G = 0.95^{T-1}$ for successful dialogs, where T is the number of system turns in the dialog. Finally, we created a set of 21 labeled dialogs, which will be used for supervised learning.

For the RNN in the HCN, we again used an LSTM with AdaDelta, this time with 32 hidden units. RL policy updates are made after each dialog. Since a simulated user was employed, we did not have real user utterances, and instead relied on context features, omitting bag-of-words and utterance embedding features.

We first evaluate RL by randomly initializing an LSTM, and begin RL optimization. After 10 RL updates, we freeze the policy, and run 500 dialogs with the user simulation to measure task completion. We repeat all of this for 100 runs, and report average performance. In addition, we also report results by initializing the LSTM using supervised learning on the training set, consisting of 1, 2, 5, or 10 dialogs sampled randomly from the training set, then running RL as described above.

Results are in Figure 4. Although RL alone can find a good policy, pre-training with just a handful of labeled dialogs improves learning speed dramatically. Additional experiments, not shown for space, found that ablating the action mask slowed training, agreeing with Williams (2008).

Finally, we conduct a further experiment where we sample 10 training dialogs, then add one to the training set just before RL dialog 0, 100, 200, ... , 900. Results are shown in Figure 4. This shows that SL dialogs can be introduced as RL is in progress – i.e., that it is possible to interleave RL and SL. This is an attractive property for practical systems: if a dialog error is spotted by a developer while RL is in progress, it is natural to add a training dialog to the training set.

7 Conclusion

This paper has introduced Hybrid Code Networks for end-to-end learning of task-oriented dialog systems. HCNs support a separation of concerns where procedural knowledge and constraints can be expressed in software, and the control flow is learned. Compared to existing end-to-end approaches, HCNs afford more developer control and require less training data, at the expense of a small amount of developer effort.

Results in this paper have explored three different dialog domains. On a public benchmark in the restaurants domain, HCNs exceeded performance of purely learned models. Results in two troubleshooting domains exceeded performance of a commercially deployed rule-based system. Finally, in a name-dialing domain, results from dialog simulation show that HCNs can also be optimized with a mixture of reinforcement and supervised learning.

References

Antoine Bordes and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *CoRR* abs/1605.07683. http://arxiv.org/abs/1605.07683.

Franois Chollet. 2015. Keras. `https://github.com/fchollet/keras`.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Proc NIPS 2014 Deep Learning and Representation Learning Workshop*.

Mihail Eric and Christopher D Manning. 2017. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. *CoRR* abs/1701.04024. https://arxiv.org/abs/1701.04024.

David Griol, Llus F. Hurtado, Encarna Segarra, and Emilio Sanchis. 2008. A statistical approach to spoken dialog systems design and evaluation. *Speech Communication* 50(8–9).

Matthew Henderson, Blaise Thomson, and Jason Williams. 2014. The second dialog state tracking challenge. In *Proc SIGdial Workshop on Discourse and Dialogue, Philadelphia, USA*.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Chiori Hori, Kiyonori Ohtake, Teruhisa Misu, Hideki Kashioka, and Satoshi Nakamura. 2009. Statistical dialog management applied to WFST-based dialog systems. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*. pages 4793–4796. https://doi.org/10.1109/ICASSP.2009.4960703.

Filip Jurčíček, Blaise Thomson, and Steve Young. 2011. Natural actor and belief critic: Reinforcement algorithm for learning parameters of dialogue systems modelled as pomdps. *ACM Transactions on Speech and Language Processing (TSLP)* 7(3):6.

Nate Kohl and Peter Stone. 2004. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. IEEE, volume 3, pages 2619–2624.

Cheongjae Lee, Sangkeun Jung, Seokhwan Kim, and Gary Geunbae Lee. 2009. Example-based dialog modeling for practical multi-domain dialog system. *Speech Communication* 51(5):466–484.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. 2000. A stochastic model of human-machine interaction for learning dialogue strategies. *IEEE Trans on Speech and Audio Processing* 8(1):11–23.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016a. A diversity-promoting objective function for neural conversation models. In *Proc HLT-NAACL, San Diego, California, USA*.

Jiwei Li, Michel Galley, Chris Brockett, Georgios Spithourakis, Jianfeng Gao, and Bill Dolan. 2016b. A persona-based neural conversation model. In *Proc Association for Computational Linguistics, Berlin, Germany*.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016c. Deep reinforcement learning for dialogue generation. In *Proc Conference on Empirical Methods in Natural Language Processing, Austin, Texas, USA*.

Lihong Li, He He, and Jason D. Williams. 2014. Temporal supervised learning for inferring a dialog policy from example conversations. In *Proc IEEE Workshop on Spoken Language Technologies (SLT), South Lake Tahoe, Nevada, USA*.

Fei Liu and Julien Perez. 2016. Gated end-to-end memory networks. *CoRR* abs/1610.04211. http://arxiv.org/abs/1610.04211.

Ryan Thomas Lowe, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. 2017. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue and Discourse* 8(1).

Yi Luan, Yangfeng Ji, and Mari Ostendorf. 2016. LSTM based conversation models. *CoRR* abs/1603.09457. http://arxiv.org/abs/1603.09457.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. Coherent dialogue with attention-based language models. *CoRR* abs/1611.06997. http://arxiv.org/abs/1611.06997.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proc Advances in Neural Information Processing Systems, Lake Tahoe, USA*. pages 3111–3119.

Min Joon Seo, Hannaneh Hajishirzi, and Ali Farhadi. 2016. Query-regression networks for machine comprehension. *CoRR* abs/1606.04582. http://arxiv.org/abs/1606.04582.

Iulian V. Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press, AAAI'16, pages 3776–3783. http://dl.acm.org/citation.cfm?id=3016387.3016435.

Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2017. A hierarchical latent variable encoder-decoder model for generating dialogues.

Lifeng Shang, Zhengdong Lu, , and Hang Li. 2015. Neural responding machine for short-text conversation. In *Proc Association for Computational Linguistics, Beijing, China*.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Satinder Singh, Diane J Litman, Michael Kearns, and Marilyn A Walker. 2002. Optimizing dialogue management with reinforcement leaning: experiments with the NJFun system. *Journal of Artificial Intelligence* 16:105–133.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Meg Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015. A neural network approach to context-sensitive generation of conversational responses. In *Proc HLT-NAACL, Denver, Colorado, USA*.

Pei-Hao Su, Milica Gašić, Nikola Mrkšić, Lina Rojas-Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Continuously learning neural dialogue management. In *arXiv preprint: 1606.02689*.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *Proc Advances in Neural Information Processing Systems (NIPS), Montreal, Canada*.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688. http://arxiv.org/abs/1605.02688.

Oriol Vinyals and Quoc Le. 2015. A neural conversational model. In *Proc ICML Deep Learning Workshop*.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve J. Young. 2016. A network-based end-to-end trainable task-oriented dialogue system. *CoRR* abs/1604.04562. http://arxiv.org/abs/1604.04562.

Jason D. Williams. 2008. The best of both worlds: Unifying conventional dialog systems and POMDPs. In *Proc Intl Conf on Spoken Language Processing (ICSLP), Brisbane, Australia*.

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

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Zhen Xu, Bingquan Liu, Baoxun Wang, Chengjie Sun, and Xiaolong Wang. 2016. Incorporating loose-structured knowledge into LSTM with recall gate for conversation modeling. *CoRR* abs/1605.05110. http://arxiv.org/abs/1605.05110.

Kaisheng Yao, Geoffrey Zweig, and Baolin Peng. 2015. Attention with intention for a neural network conversation model. In *Proc NIPS workshop on Machine Learning for Spoken Language Understanding and Interaction*.

Steve Young, Milica Gasic, Blaise Thomson, and Jason D. Williams. 2013. POMDP-based Statistical Spoken Dialogue Systems: a Review. *Proceedings of the IEEE* PP(99):1–20.

Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *CoRR* abs/1212.5701. http://arxiv.org/abs/1212.5701.

A Supplemental Material

A.1 Model implementation details

The RNN was specified using Keras version 0.3.3, with back-end computation in Theano version 0.8.0.dev0 (Theano Development Team, 2016; Chollet, 2015). The Keras model specification is given below. The input variable `obs` includes all features from Figure 1 step 6 except for the previous action (step 13) and the action mask (step 5).

```
# Given:
# obs_size, action_size, nb_hidden

g = Graph()
g.add_input(
  name=’obs’,
  input_shape=(None, obs_size)
)
g.add_input(
  name=’prev_action’,
  input_shape=(None, action_size)
)
g.add_input(
  name=’avail_actions’,
  input_shape=(None, action_size)
)
g.add_node(
  LSTM(
    n_hidden,
    return_sequences=True,
    activation=’tanh’,
  ),
  name=’h1’,
  inputs=[
    ’obs’,
    ’prev_action’,
    ’avail_actions’
  ]
)
g.add_node(
  TimeDistributedDense(
    action_size,
    activation='softmax',
  ),
  name='h2',
  input='h1'
)
g.add_node(
  Activation(
    activation=normalize,
  ),
  name='action',
  inputs=['h2','avail_actions'],
  merge_mode='mul',
  create_output=True
)
g.compile(
  optimizer=Adadelta(clipnorm=1.),
  sample_weight_modes={
    'action': 'temporal'
  },
  loss={
    'action':'categorical_crossentropy'
  }
)
```

Model sizes are given in Table 3. Example dialogs are given below for each of the 5 dialog systems. For space and readability, the entity tags that appear in the user and system sides of the dialogs have been removed – for example, *Call* `<name>`*Joan*`</name>` is shown as *Call Joan*.

A.2 bAbI Task5 example dialog

good morning
hello what can i help you with today
i'd like to book a table with italian food
i'm on it
[silence]
where should it be
in paris
how many people would be in your party
for six people please
which price range are looking for
in a cheap price range please
ok let me look into some options for you
[silence]
`api_call italian paris six cheap`
`[database results returned]`
[silence]
what do you think of this option: resto_madrid_cheap_spanish_8stars
no this does not work for me
sure let me find an other option for you
[silence]
what do you think of this option: resto_madrid_cheap_spanish_6stars
do you have something else
sure let me find an other option for you
[silence]
what do you think of this option: resto_madrid_cheap_spanish_1stars
it's perfect
great let me do the reservation
may i have the phone number of the restaurant
here it is resto_madrid_cheap_spanish_1stars_phone
thanks
is there anything i can help you with
no thank you
you're welcome

A.3 bAbI Task6 example dialog

In the system turn marked [*], the system provided venue information without first making an API call.

Hello , welcome to the Cambridge restaurant system . You can ask for restaurants by area , price range or food type . How may I help you ?
im looking for canapes food in any price range
I am sorry but there is no canapes restaurant that matches your request [*]
how about european
`api_call european R_location R_price`
`[database results returned]`
[silence]
michaelhouse_cafe is a great restaurant serving european food and it is in the expensive price range
food an
What part of town do you have in mind?
any any part of town
michaelhouse_cafe serves european food .
may i have the address and phone number please
The phone number of michaelhouse_cafe is michaelhouse_cafe_phone
thank you good bye
you are welcome

A.4 Task5 and Task6 context features

Context features used in Section 4 are in Table 4.

A.5 Forgot password example dialog

Some system actions have been shortened for space.

How do I recover my password
Okay, you don't need to remember your password, we can reset it. Would you like to try that?
Yes
To reset your password, please see this URL ... Was that helpful?
I don't have a flashdrive
Let's connect you to a person who can help you

A.6 Account access example dialog

Some system actions have been shortened for space.

cannot sign in to my email
Please check caps lock is off, and that your email is spelled correctly. Are you able to access your account now?

	bAbI Task5	**bAbI Task6**	**Reset password**	**Account access**	**Dialer**
Utterance embed.	300	300	300	300	0
Bag of words	85	523	1124	278	0
Context features	4	14	0	0	17
`obs_size`	389	837	1424	578	17
`action_size`	16	58	7	16	14
`nb_hidden`	128	128	128	128	32

Table 3: Dimensions of the 5 HCNs in this paper.

Feature	**bAbI Task5**	**bAbI Task6**
Presence of each entity in dialog state	4	3
Presence of each entity in current utterance	–	3
Whether DB has been queried yet	–	1
Whether DB results are empty	–	1
Whether DB results are non-empty	–	1
Whether any DB results have been presented	–	1
Whether all DB results have been presented	–	1
Whether any DB results are available to present	–	1
Whether curent query yields no results in training set	–	1
Whether cuisine in current query yields no results in training set	–	1
Total context features	4	14

Table 4: Binary context features used to convey entity and database state in Section 4.

no
If you still can't sign in, please go the the following page and check that the email server is running ok... Is everything up and running?
yes
You need to reset your password - here's how ... Did this solve your problem?
still cannot get in
Did you receive the security code?
yes
Check you entered the right numbers. Did this solve your problem?
no
Have you seen a message that your profile is temporarily blocked?
just your message of update on sept 15
Did you turn on two-step verification recently?
whats that?
Sorry, I can't help. Rephrase your question, or click below to talk to a person.

A.7 Name dialing example dialogs

How can I help you?
Call Joe
There's more than one person named Joe. Can you say their full name?
Call Joe Adamson
`SavePhonetypeavail()`
Calling Joe Adamson, work

`PlaceCall()`

How can I help you?
Call Sally on her home
Sorry, I don't have a home number for Sally Smith. I only have a work phone. Do you want to call that number?
no
Oh, sorry about that. Goodbye.