CS671A: Project Report
Task Oriented Dialog Systems

Group - 4
Abhishek Verma[14026] , Aniket Maiti[14096] , Parth Sharma[14449] , Raghvendra Kushwaha[13817527] , Rahul Kumar[150548] , Shivam Taji[14745]

[1] **Code link:** https://github.com/abhi608/task_oriented_nlp_system

{abhivm,aniketmt,parth,raghv,krrahul,shivamt}@iitk.ac.in

Abstract. *Teaching machines to converse naturally with humans to achieve specific tasks is very challenging. Currently developing task oriented dialogue systems require specific hand-crafting. Also obtaining the dataset may be very cumbersome and costly. As a result, it is very difficult to scale them to new dimensions.The first implementation targets the first problem of specific hand-crafting. The second implementation improves it further by considerably reducing the data required. We show that end-to-end models can achieve good results in case of goal oriented dialogues.*

1. Introduction

Task oriented dialogue systems are used for achieving a particular goal using natural language. Specific tasks may include restaurant reservation, getting technical support or placing a phone call and many such services.

Building such systems are difficult because they are application specific and there is limited availability of training data. Although end to end models do fairly good in non-goal oriented settings, it is important to test if they can do well in goal-oriented settings or not. In this project, we have implemented two end to end goal oriented dialogue systems and compared their performances. The first implementation has an attention based architecture called memory networks. In this case, the tasks have been broken down into several sub-tasks. The second implementation introduces Hybrid Code Networks(HCN)[3] which combines an RNN with domain-specific knowledge encoded as software and system action templates.

2. Related Work

Modelling conversation as partially observable Markov decision process is the most successful strategy for goal-oriented dialogues[7]. However, they require many hand-crafted features for the state and action state representation.There are two lines of work, in task based dialog systems, braodly speaking. The first consists of breaking down the dialog system into a sequence of components creating a pipeline, typically including language understanding, dialog state tracking, action selection policy, and language generation.[8]. Recent papers have implemented the policy as feed-forward neural networks trained with supervised learning followed by reinforcement learning. [6]
The second approach utilizes recurrent neural networks (RNNs) enabling end-to-end learning which map from an observable dialog history directly to a sequence of output words.[4]. With the addition of API call actions, these systems can be applied to task-oriented domains, learning an RNN using Memory Networks, [5], copy-augmented networks.[2] etc, to name a few.

3. Dataset description

The dataset is based on a knowledge base which contains the restaurants that can be booked and their properties. Each restaurant has 4 properties, namely, type, location, price range and rating along with their addresses and phone numbers being listed in the knowledge base. The knowledge base is queried using API calls which returns a list of facts related to the restaurant.Dialogues are created when a user makes a request for a particular kind of restaurant. The user can say things in different ways but the bot always uses the same pattern. The data is collected using a simulator where users interact with chatbot and then rate each and every action taken it.

4. Models Studied

4.1. Memory Network

Memory Networks[5] are a recent class of models that have been applied widely to build Task Oriented Dialog Systems. We provide a story (sequence of sentences) along with a query (single sentence) as input to the model and it predicts the action as well as reply to that query. This model stores the whole story in memory and then iteratively reads from that memory to predict response to the query.

4.1.1. Approach and Training

This model[1] takes a story (**s** $\{x_1, x_2 ..., x_n\}$) and a query(**q**) as input and predicts the output response(**a**). The training steps for this model are as follow:

- **Input memory representation**: We are given an input set $\{x_1, .., x_n\}$ to be stored in memory. The entire set of $\{x_i\}$ is converted into memory vectors $\{m_i\}$ of dimension *d* computed by embedding each x_i in a continuous space using an embedding matrix *A* (of size d x sentence-size). The query *q* is also embedded (using another embedding matrix *B* with the same dimensions as *A*) to obtain an internal state *u*. In the embedding space, we compute the match between *u* and each memory m_i by taking the inner product followed by a softmax:

$$p_i = Softmax(u^T m_i)$$

- **Output memory representation**: Each x_i has a corresponding output vector c_i (given by another embedding matrix C). The response vector from the memory *o* is then a sum over the transformed inputs c_i, weighted by the probability vector from the input:

$$o = \sum_j p_i c_i,$$

- **Generating the final prediction**: In the single layer case, the sum of the output vector *o* and the input embedding *u* is then passed through a final weight matrix *W* (of size V x d) and a softmax to produce the predicted label:

$$\hat{a} = softmax(W(o + u))$$

  Now we extend our model to handle K-hops operation:

- The sum of output vector o_k and input vector u_k becomes the input vector for next $(k + 1)^{th}$ hop operation:

$$u^{k+1} = o^k + u^k$$

- For each layer embedding matrix A^k, B^k are used to find input memory vector mi and output vector u^k.
- In the final layer we predict the answer $a^k = softmax(Wu^{K+1})$

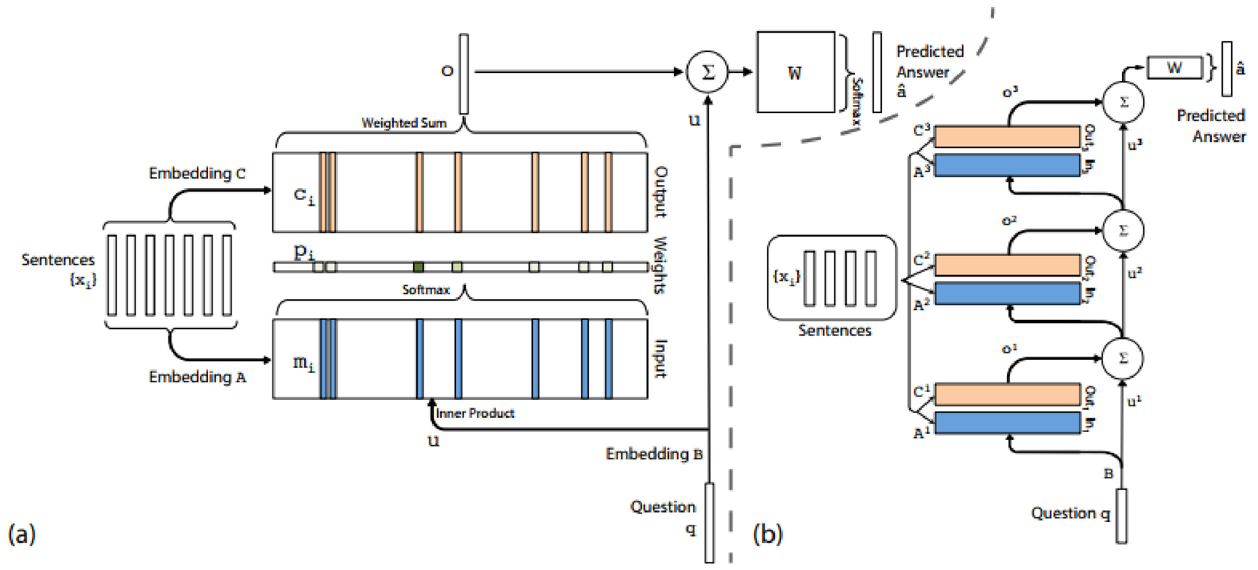


Figure 1. Architecture of Memory Networks

Weight tying is implemented in two ways:

1. Using a RNN-like architecture. In this architecture, the input and output embeddings are the same across different layers, i.e. $A_1 = A_2 = ... = A_K$ and $C_1 = C_2 = ... = C_K$. A linear mapping H is added to the update of u between hops, i.e.,

$$u^{k+1} = o^k + Hu^k$$

2. Using a adjacent lateral structure where $A_k = C_{k-1}$ and $A_0 = B$. This structure was also implemented, but gave worse results than the RNN-like structure.

4.2. Hybrid Code Network(HCN)

End-to-End learning RNN models for dialog systems performs very well but require large amounts of data for training. This method combines an RNN with domain specific knowledge encoded as software and system action templates. HCN reduces the training data required and the same retains the benefit of inferring a latent representation of a dialog state. Task oriented dialogue systems can be used to accomplish goals using natural language but the dependencies between modules introduce considerable complexity. Also training each module requires specific labels.Also, end-to-end methods lack a general mechanism for injecting domain knowledge and constraints.

HCN solves these problems by learning an RNN and allowing a developer to express domain knowledge via software and action templates with much less data.

4.2.1. Approach and Training

- Combines an RNN with domain-specific knowledge encoded as software and system action templates
- HCNs considerably reduce the amount of training data required, while retaining the key benefit of inferring a latent representation of dialog state.
- HCNs can be optimized with supervised learning, reinforcement learning,or a mixture of both.
- 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.

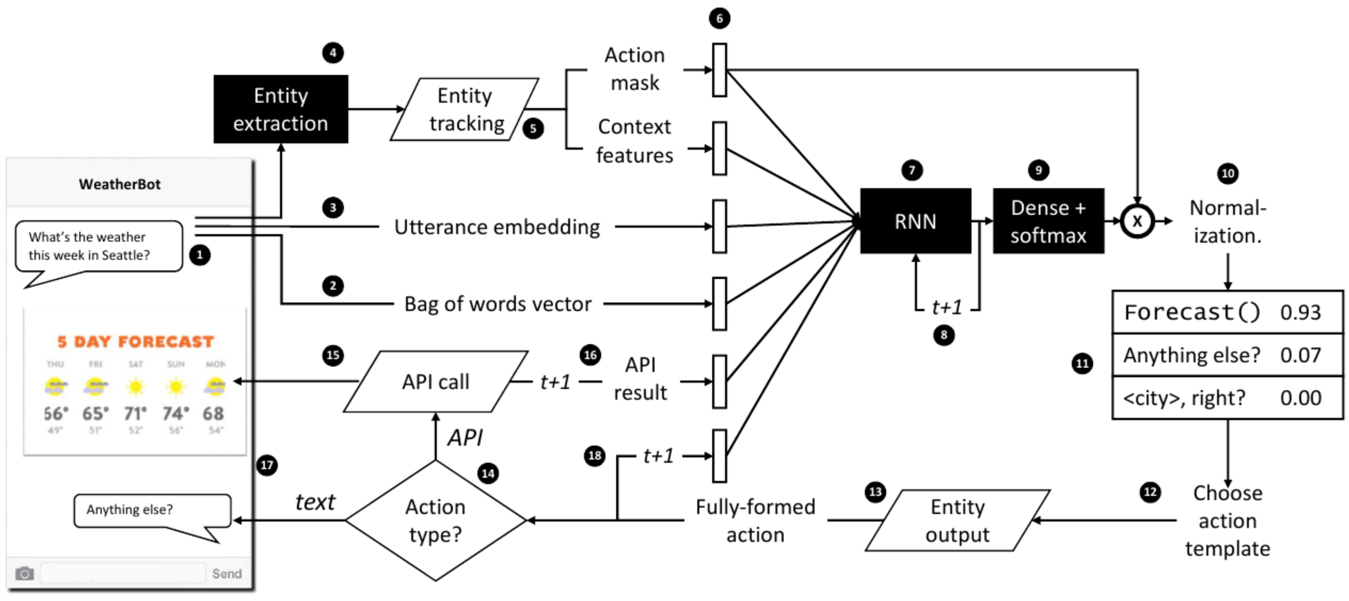


Figure 2. Architecture of Hybrid Code Networks

- Both the RNN and the developer code maintain state. Each action template can be a textual communicative action or an API call.

First, we extract features from the User Input:-

- Step 1: User provides the utterance.
- Step 2: A bag of words vector is formed
- Step 3: Above vector is converted into an utterance embedding using a pre-trained embedding model
- Step 4: Entities are extracted
- Step 5: The text and entities are fed to an entity tracker
- Step 6: The feature components from steps 1-5 are concatenated to form a feature vector
- Step 7: This vector is fed to an LSTM

Now, the state tracking happens implicitly using an RNN(LSTM):

- Step 8: The RNN computes a hidden state vector, which is retained for the next time step
- Step 9: The above hidden state vector is passed to a dense layer with a soft-max activation, with output dimension equal to the number of distinct system action templates. Thus, the output of this step is a distribution over action templates.
- Step 10: The action mask is applied as an element-wise multiplication, and the result is normalized back to a probability distribution
- Step 11: The above step forces non-permitted actions to take on probability zero.
- Step 12: From the distribution obtained in Step 11, an action is selected

Once, we have obtained a distribution over the actions. We can either select the max action (if deployed/ in supervised learning mode) or sample an action from the given distribution (if training with reinforcement learning so as to have exploration)

- Step 13: The selected action is then passed to "Entity output" developer code that can substitute in entities. This produces a fully-formed action.
- Step 14 and following steps, 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.

- Step 16: APIs can act as sensors and return features relevant to the dialog,so these can be added to the feature vector in the next time-step
- Step 17: If the action is text, it is rendered to the user and cycle then repeats.
- Step 18: The action taken is provided as a feature to the RNN in the next time-step.

4.3. A Network-based End-to-End Trainable Task-oriented Dialogue System

The authors focus on the problem on task-oriented dialog systems. While end-to-end learning dialog systems for non-goal-oriented chats was demonstrated by Vinyals et al[**?**] and Serban et al [4] ,task-oriented dialog systems usually require multiple components which need to be separately trained. They demonstrate an end-to-end neural network based goal-oriented dialog system.

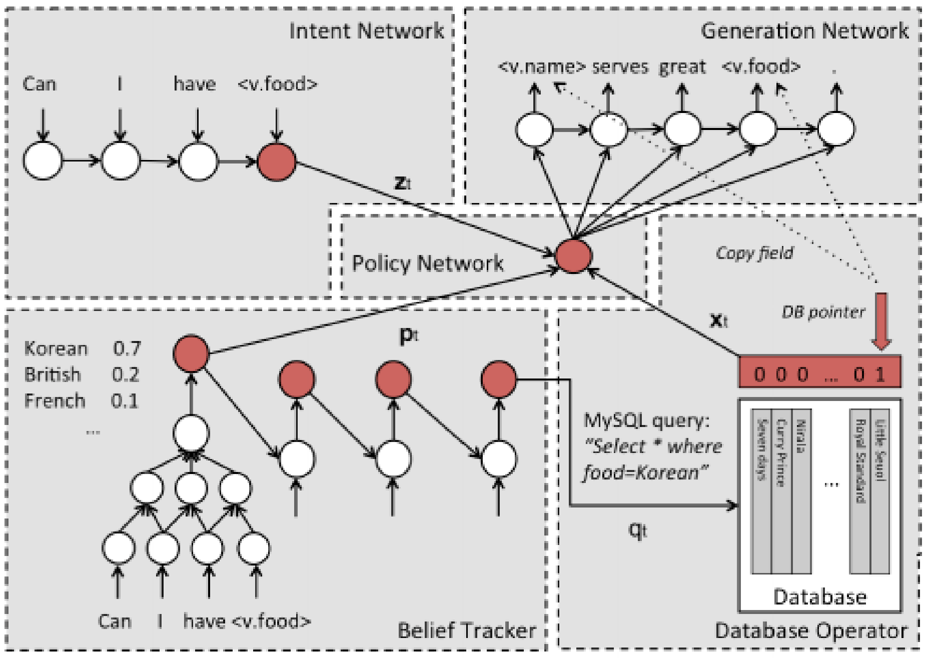


Figure 3. The end-to-end trainable dialogue system framework proposed by Wen et al [6]

4.3.1. Approach and Training

- **Intent Network :** The Intent Network encodes the incoming sentences into a vector z_t in turn t. The authors investigate the use of both LSTM and CNN in performing this encoding.

$$\mathbf{z_t} = \mathbf{z_t^N} = \text{LSTM}(w_0^t, w_1^t, \dots, w_N^t)$$
$$\mathbf{z_t} = \text{CNN}(w_0^t, w_1^t, \dots, w_N^t)$$

- **Belief Trackers :** Belief tracking is the process of learning a distribution of user's current goals in the conversation. The authors use an RNN with a CNN feature extractor for each slot
- **Database Operator :** The Database operator takes the output of the belief tracker and extracts the relevant information from database for the query slot.
- **Policy Network:** The policy network takes in as inputs the output from the intent network $\mathbf{z_t}$, the belief state $\mathbf{p_s^t}$ and the DB truth value vector $\mathbf{x_t}$ and produces the output vector $\mathbf{o_t}$ as follows

$$\mathbf{o_t} = tanh(\mathbf{W_{zo}z_t} + \mathbf{W_{po}\hat{p}_t} + \mathbf{W_{xo}}\hat{x}_t \qquad (1)$$

- **Generation Network:** It takes in the o_t to produce an output sentence an LSTM based language generator [?]. The sentences is generated with generic tokens for slots which are then replaced by their values.

$$P(w_{j+1}^t | w_j^t, \mathbf{h}_{j-1}^t, \mathbf{o_t}) = LSTM_j(w_j^t, h_{j-1}^t, \mathbf{o_t}) \tag{2}$$

Due to fuzzy details in the paper, slightly unreadable available code and our lack of knowledge of Theano, we weren't able to implement this paper on our own.

5. Results & Observations

5.1. Memory Networks

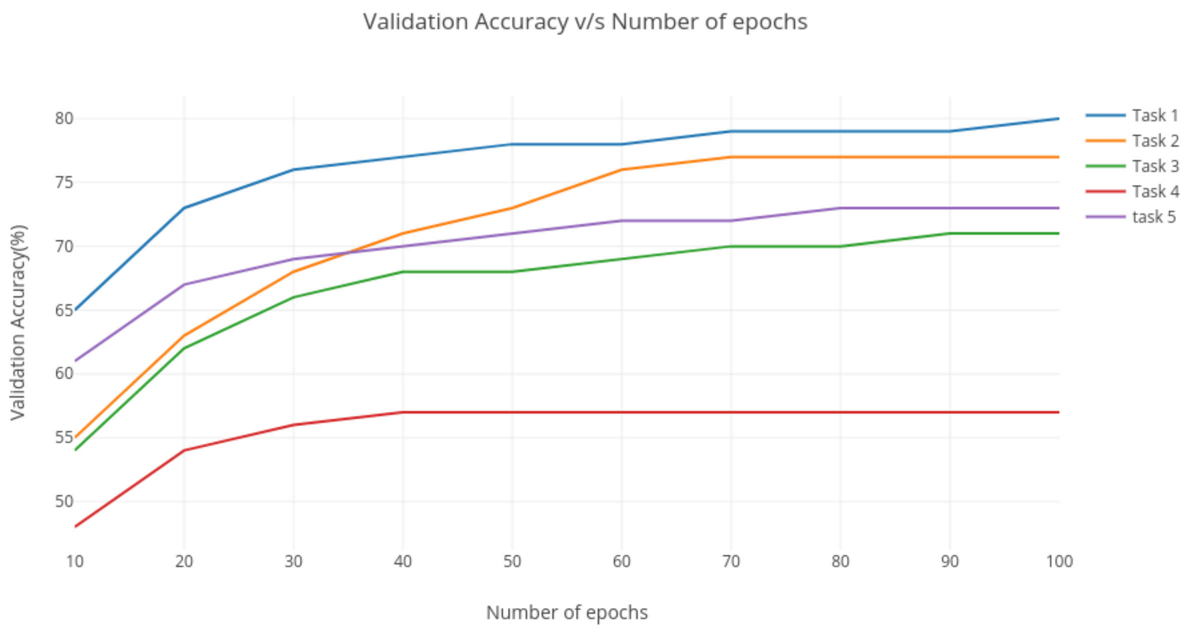
We experimented with various optimizers with training for 100 epochs for each task. Some observations are as follow:

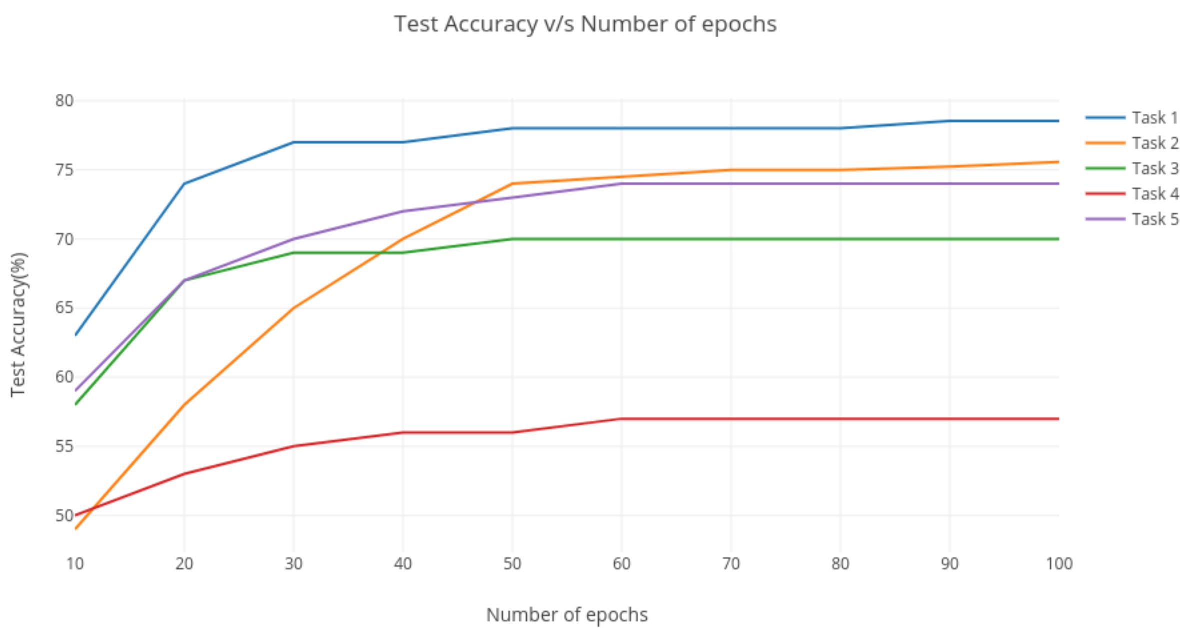
- When we used the SGD optimizer with momentum, the train error decreased rapidly initially but started increasing after about 10 epochs.
- When RMSprop optimizer with momentum was used, the rate of decrease of error was very high, though after around 8 epochs the error stated increasing rapidly.
- Adam optimizer worked best in this case. Though the error decreased slowly, but the minima was hit using this optimizer and it gave some good accuracies.
- Adaptive learning rate performed better than constant learning case. Basically, error at each epoch was monitored and the learning rate was decresed by a factor of 0.1 when the error became almost constant.

We got best results with Adam optimizer. The results for Adam optimizers are shown below:

Table 1. Accuracy

Task	Validation Accuracy (%)	Test Accuracy (%)
1	80.12	78.54
2	77.56	75.56
3	70.85	70.34
4	56.67	57.13
5	73.63	74.12





5.2. Hybrid Code Networks

We experimented with various optimizers with 80 epochs (compared to 12 in the original paper).

- The results with Vanilla SGD were very poor(43% acc after 40 epochs). This is reason that we've ignored them in the graphs(because of scaling).
- It should also be noted the HCN only takes in 200 conversations as examples instead of the Memory Networks paper which uses 1600 conversations to achieve comparable accuracy. This is attributed to the simplification applied due to domain specific software.
- In the original implementation, adadelta was performing better than adagrad. Adagrad was also better in the tensorflow implementation. But in case of our pytorch implementation, adagrad proved to be better than adadelta as can be seen in the graphs.

The accuracies of various optimizers on 50 data samples taken from *BaBi* dataset which were not present in train dataset are as follow:

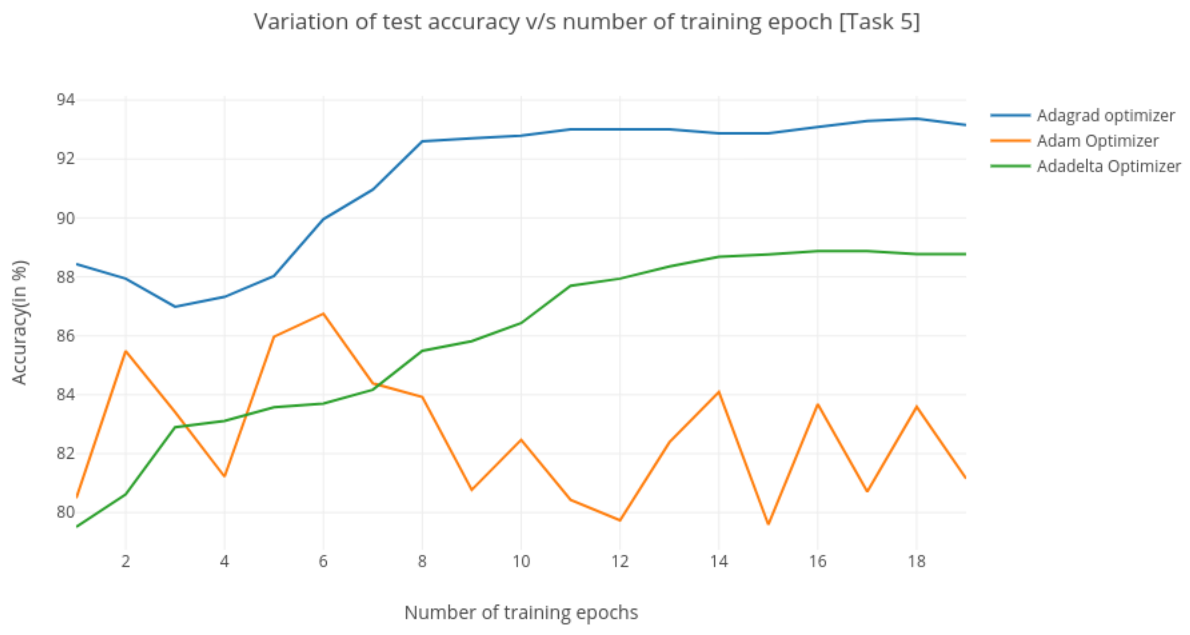
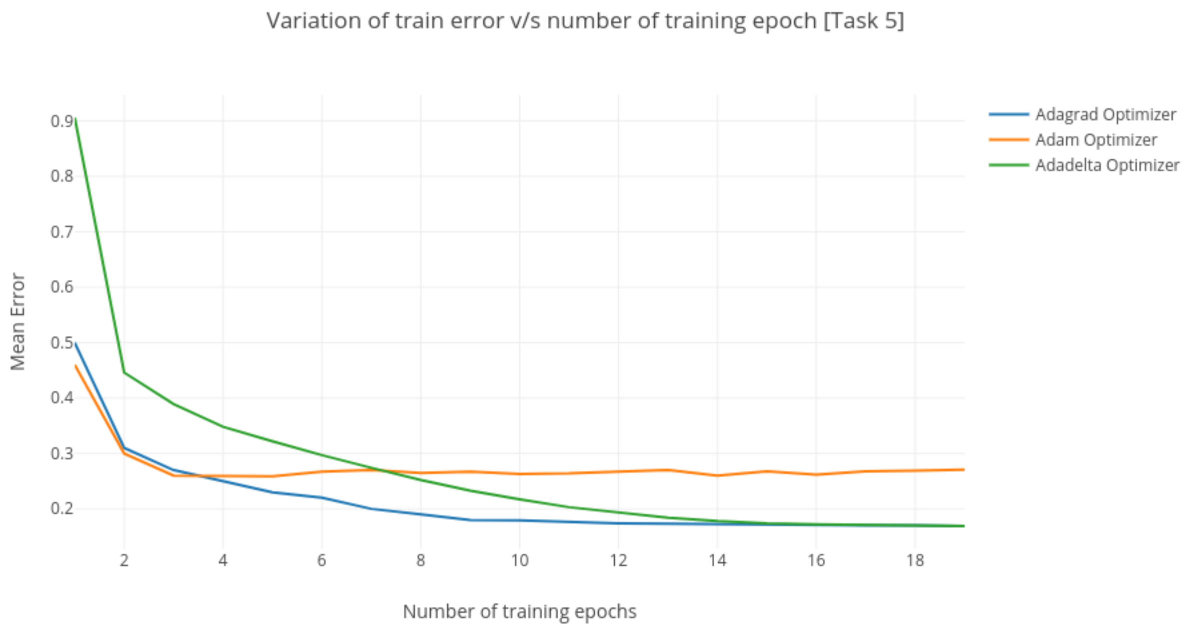


Table 2. Accuracy with different optimizers

Model	Adagrad Optimizer	Adam Optimizer	Adadelta Optimizer
HCN [Task 5]	93.68%	86.53%	89.13%



6. Future Work

The above mentioned models can be scaled to large domains which we hope to pursue in our future work.

Currently, memory networks do not perform Natural Language generation. We aim to implement a module which achieve the same so that the training process will no more be a multiclass classification(since the number of classes is very high, classification needs a lot of training data). This will reduce the amount of data required to train the model. Also, this will make the error converge in lesser epochs.

HCNs can be extended by incorporating lines of existing work, such as integrating the entity extraction step into the neural network adding richer utterance embeddings and supporting text generation. We can also explore using HCNs with automatic speech recognition (ASR) input.

7. Contributions

Table 3. Contribution

Name	Contibution (%)
Abhishek Verma	20.00
Parth Sharma	20.00
Aniket Maiti	20.00
Rahul Kumar	20.00
Shivam Taji	10.00
Raghvendra Kushwaha	10.00

References

[1] Y-Lan Boureau Jason Weston Antoine Bordes. Learning end-to-end goal-oriented dialog. *arXiv:1605.07683*, 2017.

[2] Mihail Eric and Christopher D Manning. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. *arXiv preprint arXiv:1701.04024*, 2017.

[3] John F Kelley. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. *arXiv:1605.07683*, 2017.

[4] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*, volume 16, pages 3776–3784, 2016.

[5] Szlam-A. Weston J. Sukhbaatar, S. and R. Fergus. End to end memory networks. *Proceedings of NIPS*, 2015.

[6] Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

[7] Gasic M. Thomson B. Young, S. and J. D. Williams. omdp-based statistical spoken dialog systems:a review. 2013.

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