Pretrain-Finetune Based Training of Task-Oriented Dialogue Systems in a Real-World Setting

Manisha Srivastava
Amazon Inc.
Seattle, USA
mansri@amazon

Yichao Lu
Amazon Inc.
Seattle, USA
yichaolu@amazon

Riley Peschon
Amazon Inc.
Seattle, USA
peschon@amazon

Chenyang Li
Amazon Inc.
Seattle, USA
cli@amazon

Abstract

One main challenge in building task-oriented dialogue systems is the limited amount of supervised training data available. In this work, we present a method for training retrieval-based dialogue systems using a small amount of high-quality, annotated data and a larger, unlabeled dataset. We show that pretraining using unlabeled data can bring better model performance with a 31% boost in Recall@1 compared with no pretraining. The proposed finetuning technique based on a small amount of high-quality, annotated data resulted in 26% offline and 33% online performance improvement in Recall@1 over the pretrained model. The model is deployed in an agent-support application and evaluated on live customer service contacts, providing additional insights into the real-world implications compared with most other publications in the domain often using asynchronous transcripts (e.g. Reddit data). The high performance of 74% Recall@1 shown in the customer service example demonstrates the effectiveness of this pretrain-finetune approach in dealing with the limited supervised data challenge.

1 Introduction

Retrieval-based dialogue systems are popular in task-oriented domains. A typical retrieval-based system encodes the dialogue context and a large set of candidate responses (templates) in a joint semantic space, and then scores how appropriate each candidate is given the dialogue context; the template with the highest score is selected as the response. These systems can use a sequence-to-sequence model (Kannan et al., 2016) or a dual-encoder style architecture (Lu et al., 2017; Lowe et al., 2015) to encode and score the context-response pair.

One major challenge for any task-oriented dialogue system is the scarcity of training data. High-quality data with all the required annotations are needed to train an accurate model. Such datasets are not readily available, and collecting them is a costly and labor-intensive process. A few synthetic datasets (Weston et al., 2015; Asri et al., 2017; Budzianowski et al., 2018) have been proposed but they do not capture the real-world variations and subtleties of the task-oriented dialogues. The limited amount of supervised training data available makes it difficult to train these models from scratch.

To overcome the issue of limited training data, the idea of finetuning a pretrained model has become a popular approach in other domains like computer vision and is recently gaining popularity in the natural language processing (NLP) domain. Pretrained models in NLP such as ELMo (Peters et al., 2018), OpenAI GPT (Radford et al., 2018), and BERT (Devlin et al., 2018) have attracted a lot of attention and achieved state-of-the-art accuracy in multiple natural language understanding tasks. In this paper, we present a methodology for training retrieval-based dialogue systems using a small amount of supervised data and a large, low-quality, unannotated dataset.

1. We demonstrate that finetuning a model (Lu et al., 2019) pretrained using the unannotated dataset performs better than directly finetuning on the clean, annotated data.
2. We experiment with different finetuning loss functions and show that a ranking based loss function performs better than classification loss for template-retrieval based dialogue systems.
3. We deploy the finetuned model in an agent assistance application for customer service, and present real-world results on live customer contacts.

In the sections that follow, we describe the data from the customer service domain that is used for pretraining and finetuning the model in section 2. In section 3, we explain how we pretrain the model

Proceedings of NAACL HLT 2021: IndustryTrack Papers, pages 34–40
June 6–11, 2021.

Raw text:
Customer: I want to cancel the shoes I ordered yesterday.
Agent: Welcome to Customer Service.
Agent: I am here to help you.
Agent: Give me a moment to look into this.

Training Sample:
Context: **CUSTOMERSTART** I want to cancel the shoes I ordered yesterday. **AGENTSTART** Welcome to Customer Service. **AGENTSTART** I am here to help you. **PROFILESTART** cancellable, carrier, membership-status. **Response**: Give me a moment to look into this.
Label: Positive

Figure 1: Training sample creation process. Given a chat transcript and profile features, a training sample is created by appending the dialogue turns and profile features. True agent response is used to create positive samples and random agent responses are used to create negative samples.

using the next-turn prediction task along with the results. Next, we present the proposed finetuning strategy, and the associated experimental setup and the results. In section 5, we present the real-world results of the deployed model. Section 6 describes the conclusion and direction for future work.

2 Data

In this work, we use data from the customer service domain —customer service chats handled in English. When customers contact customer service regarding their issue (e.g., order tracking, payment questions), the routing system connects the customer to an agent based on the specific issue type. Agents resolving customer issues have access to a wide variety of profile information (e.g. customer details, order status, and internal APIs) to execute actions such as canceling or refunding an order. For our experiments, we select a delivery-related customer issue. In the following subsection, we explain how we collect the pretraining and finetuning data.

2.1 Pretraining Data

The pretraining data include historical customer service chat transcripts for a delivery-related issue. It is important to note these transcripts only contain the dialogue turns. Contextual information (e.g. customer profile, order details, actions executed by the agents) is either missing or inaccurate. The pretraining dataset consists of a few hundred thousand chats (see Table 1). The conversation in these chat transcripts can exhibit high variability, despite following the same customer issue, due to policy changes, unconstrained conversations like side talks, and agent locale variability.

Table 1: Training data statistics.

DATASETS	PRETRAIN		FINETUNE	
	TRAINING	TEST	TRAINING	TEST
NUMBER OF CHATS	382,688	2000	6366	400
NUMBER OF AGENTS TURNS	8045059	4498	65908	3188

Figure 1 shows part of a chat transcript and how it is processed to create the training data. Each agent turn in the transcript is converted into a training sample. To create the dialogue context, previous turns in the conversation history, prior to the current agent turn, are prepended by a special token to indicate whether it is an agent or customer turn. A separate token is used to distinguish profile features (e.g. customer's profile, order details) from the dialogue turns. As explained in section 3, pretraining is done using next sentence prediction task and so it requires positive and negative context response pairs. To create the pretraining dataset, true agent responses create a positive pair, while random agent responses create negative pairs. We also use the incomplete and noisy profile information that is available without any human annotation. We call this **pretrain training dataset**.

2.2 Finetuning Data

The large pretraining dataset is not collected in a standardized manner. As a result, said dataset is noisy —there are inconsistencies in chat dialogues and profile information (e.g. order details, customer profile, and actions) is missing and inaccurate. The finetuning dataset, in contrast, is collected in a controlled manner using specialist agents to ensure accurate and complete annotations.

The finetuning data consists of a few thousand chats for the selected delivery issue, collected over a period of 2 months, handled by a group of 20 specialist agents. These chats have all relevant annotations (customer profile and order details) for each dialogue turn. Agents are instructed to choose the response from a template pool as much as possible, free-typing only if the response does not exist in the template pool. The pool consists of 85 template responses. These responses are extracted from historical chat transcripts and cover the most common delivery-related use cases. The specialist agents are trained to handle contacts in a constrained and consistent manner without sacrificing the customer experience. For example, they are trained to drive

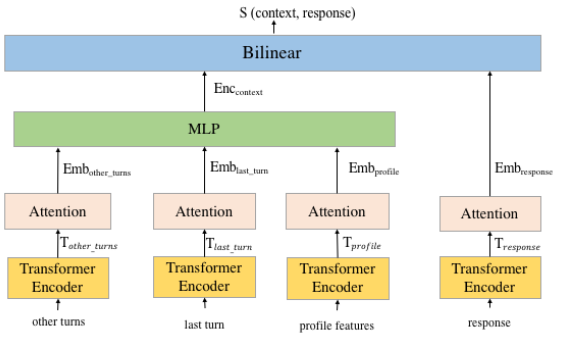

Figure 2: Pretraining model architecture. Separate transformer encoders are used to encode the dialogue history (last turn, other turns), profile features, and response. The encoded dialogue turns and profile are passed through MLP to get the encoded context. The encoded response and context are passed through bilinear layer to get the final score of the pair.

the conversation towards the solution and avoid side conversations. To ensure consistency, we instituted general rules to the agents on how and when to greet, apologize, make policy exceptions, provide reassurance, etc. The agent training ensured customers are not adversely affected in the process of this constrained data collection.

The collected chats are processed in the same way as shown in Figure 1 and explained in the last section. The dataset, referred to as **finetune training dataset**, is generated in the same way as pretraining data with one caveat: for each conversation context, negative samples are generated using templates scored high by the pretrained model, as opposed to random sampling unrelated agent responses. The number of negative samples is selected using cross-validation.

2.3 Evaluation Data

We have two evaluation datasets – **pretrain test** and **finetune test**. To create the pretrain test dataset, we use historical chat transcripts from a period that does not overlap with the pretrain training dataset. Each test sample includes conversation context (previous turns and extracted profile features) and random responses (including the true agent's response). During evaluation, the trained model is used to rank the responses for each dialogue context. Similarly, to create the finetune test dataset, we use the dataset collected from the specialized agents. For each dialogue context, we store the template response selected by the agent as positive and all other template responses as negative.

Table 1 shows the statistics for each of the pretrain and finetune datasets. During the evaluation, the model ranks all responses for a given dialogue context. We use Recall@1 as our evaluation metric, which measures how many times the correct response was ranked at the top by the model. We also report MRR (mean reciprocal rank), which is the harmonic mean of the rank of the correct response.

3 Pretraining

In this section, we introduce the next sentence prediction based pretraining used to pretrain the model.

3.1 Model

Our binarized next sentence prediction pretraining is effectively a classification task, classifying a pair of conversation context and agent response as positive (appropriate) or negative (not appropriate). The input to the pretraining model is the context (C) response (R) pair where the conversation context includes dialogue turns (last turn, other turns) and profile features. The pretraining model is similar to (Lu et al., 2019) as the response ranking model uses multiple transformer-based (Vaswani et al., 2017) encoders to encode different parts of the context and the response.

The context is encoded using three transformer encoders (Figure 2) that separately encode the profile features, last-turn, and all other turns in the context; see equation 1, 2, 3. Dot Product Attention (Luong et al., 2015) is applied to the transformer outputs. The transformer outputs are the key and value, while separate query vectors are learned for each output. Each query vector is randomly initialized and trained like other model parameters. The encoded last turn, profile features, and other turns are passed through a Multi-Layer Perceptron to get the encoded context (Enc_C). The response encoding (Emb_R) is also obtained using equation 2 and 3.

$$Enc_C = MLP(Emb_{other\_turns} | Emb_{last\_turn} | Emb_{profile}) \quad (1)$$

$$Emb_x = Attention(T_x, q_x) \quad (2)$$

$$T_x = Transformer(emb_x) \quad (3)$$

where | is the concatenation, emb_x is a vector of size $e \times n$: e is the embedding dimension, n is the number of words; T_x is a vector of size $h \times n$: h is

Table 2: We present MRR and Recall@1 of the finetuned models on the finetune test dataset. The baseline 'No pretraining'is a model without any pretraining, 'Pretrained mode'is pretrained on pretrain training dataset.

MODELS	MRR (%)	RECALL@1(%)
NO PRETRAINING($M_{baseline}$)	32.8	23.2
PRETRAINED MODEL(M_{tuned})	60.6	54.6

the hidden size; q_x is a query vector of dimension h which is initialized randomly and trained along with other parameters. Emb_x and Enc_C are both vectors of dimension h. Enc_C and Emb_R are then passed through a bilinear layer that outputs a probability score grading how appropriate the candidate response is given the context; see 4 and 5.

$$P[(y_t = +1)|(C, R)] = Sigmoid(S) \quad (4)$$

$$S = Enc_C \cdot Emb_R \quad (5)$$

where $y_t \in \{0, 1\}$ is the label of context response pair; $P[(y_t = +1)|(C, R)]$ is the probability that the context response pair is positive; $\cdot$ is the dot product operator. Since we treat this as a classification problem, we use binary cross-entropy loss for training.

3.2 Training Setup

We train the pretrained model (M) using the pretrain training dataset as described in Section 2. We use the transformer implementation provided by MXNet Gluon NLP [1]. We use 4 encoder layers, 4 heads in multi-head attention, hidden size of 512 and vocabulary size of 10K. The maximum sequence length of the context is 180 tokens, last turn is 35 tokens, profile feature is 6 tokens, and response is 35 tokens. We train the model using binary cross-entropy loss and stop when the validation MRR and Recall@1 start dropping.

We finetune the pretrained model on the finetune training dataset and evaluate it on the finetune test dataset. To finetune the model, we initialize the model M with the pretrained model parameters and run a few epochs on the finetune training dataset, stopping when the validation performance begins dropping. Let's call this finetuned model M_{tuned}. We also train a baseline model ($M_{baseline}$) that is another model like M but trained directly on the small finetune training dataset.

[1] https://gluon-nlp.mxnet.io/

3.3 Results

Table 2 shows the MRR and Recall@1 on the finetune test dataset. M_{tuned} demonstrates an average improvement of 31.4% on Recall@1 and 28% on MRR compared to $M_{baseline}$. These results show that pretraining on a large, unannotated dataset can give significant performance boost over a model with no pretraining.

4 Finetuning

Simple finetuning using the small finetune training dataset can lead to overfitting. In this section, we describe our finetuning approach, which incorporates regularization to avoid overfitting and the loss function that better caters to the task of template ranking.

4.1 Model

Due to the limited amount of finetune training data available, simple finetuning M can lead to overfitting —forgetting knowledge acquired during pretraining. As shown in Table 3, simple finetuning M on the finetune training dataset degrades the performance of the model on the pretrain test data significantly. In order to prevent the model from forgetting, both M and M_{tuned} should have similar performance on the pretrain test data.

4.1.1 Regularization

To prevent the forgetting issue, a fraction of the pretrain training dataset is mixed with every batch of the finetuning training dataset (He et al., 2019). During each gradient descent step of finetuning, one gradient step is taken on the finetune data batch, with another gradient step on the pretrain data batch.

4.1.2 Training loss

During pretraining, we use binary cross-entropy loss (L_{BCE}), classifying each context response pair as positive or negative. Given context response pairs and the corresponding labels, cross-entropy loss can be calculated as follows:

$$L_{BCE} = -\sum_{t=1}^{n} y_t * log(S(C, R)) + (1 - y_t) * log(1 - S(C, R)) \quad (6)$$

where $y_t \in 0, 1$ is the label of the context response pair (C, R); $S(C, R)$ is calculated using equation 5. L_{BCE} is limited by its inability to capture the relative score of the templates —essentially the

Table 3: The performance of model M finetuned on the finetune training dataset with and without regularization. We show that data-mix regularization is effective in maintaining the performance of the model on the pretrain test dataset.

DATASETS	PRETRAIN TEST DATASET		FINETUNE TEST DATASET	
	MRR(%)	RECALL@1(%)	MRR (%)	RECALL@1 (%)
M(NO FINETUNING)	80.3	76.9	41.3	32.8
M_{tuned} WITH NO REGULARIZATION	33.7	22.5	60.6	54.6
M_{tuned} USING DATA-MIXING REGULARIZATION	78.7	75.1	60.7	54.5

base logic for ranking of templates. This is critical for retrieval-based dialogue systems because, for each context, all templates receive a ranking and the top-ranked template from the pool is selected.

Hence, for finetuning, we chose a new loss function to incorporate relative scores of templates similar to (Henderson et al., 2017). We directly minimize the negative log probability of the true response given the context as shown below:

$$L_{ranking} = -log(P(R/C)) \propto \frac{e^{S(C,R)}}{\sum_{i=1}^{n} c^{S(C,R_i}} \tag{7}$$

where C is the context; R is the true agent response; $P(R|C)$ is the probability of the true response given the context; R_i is i^th response; n is all possible responses; $S(C, R)$ is the score of a pair of context and response calculated using equation 5. Instead of normalizing over all R_i, we sample 10 responses from the template pool (including the correct response). This new loss function better represents the ranking problem.

4.2 Experimental Setup

For finetuning the model, we initialize the model with the pretrained model parameters and finetune all layers using the finetune training dataset.

4.3 Results

In this section, we study the effect of regularization and the different loss functions on finetuning.

4.3.1 Regularization

The effect of regularization during finetuning is summarized in Table 3. We show the MRR and Recall@1 metrics on both the pretrain test and finetune test datasets. As a baseline, we don't finetune the model (M) at all, but directly evaluate the pretrained model on both the test datasets. From the results, we see that data-mix regularization maintains the performance of the model on the pretrain test dataset.

Table 4: The performance of the finetuned model on the finetune test dataset, the model trained using ranking loss outperforms the cross-entropy loss based finetuned model.

	MRR (%)	RECALL@1(%)
CROSS-ENTROPY LOSS	60.7	54.5
RANKING LOSS	63.9	58.3

4.3.2 Ranking Loss

Table 4 shows the result of changing the loss function from L_{BCE} to $L_{ranking}$. The result shows the effect of different loss functions on M finetuned using mix data regularization; the same effect is seen with other regularization strategies. Changing the loss function to $L_{ranking}$ improves the performance of the model by 3% on MRR and 4% on Recall@1 on the finetune test dataset.

5 Online Evaluation

5.1 Setup

We deployed the proposed model in a customer service agent-support application that agents use to resolve live customer contacts. The model is deployed as a service using Amazon Sagemaker [2]. The agent-support application calls the Sagemaker endpoint with the current context (previous dialogue turns and profile information), and the model returns the highest scored response from the template pool. The proposed model is able to recommend responses without any significant latency impact on the overall application.

The agent-support application presents the agents with the standard chat interface, except it replaces the text box with the top-suggested response from the model. Every time the customer or the agent enters text into their chat window, the model refreshes the response recommendations for the next agent utterance. The agents can accept or reject the model's response recommendation. If they reject the model's recommendation, they can type on their own. Since the model is deployed in a human-in-the-loop setup, we use it to evaluate the performance of the pretrained and finetuned

[2]https://docs.aws.amazon.com/sagemaker/

Table 5: The Recall@1 and % contacts resolved of pretrained and two finetuned models using cross-entropy and ranking loss respectively on live customers. M is the pretrained model; M_{tuned} is finetuned model.

	M	M_{tuned} WITH L_{BCE}	M_{tuned} WITH $L_{ranking}$
NUMBER OF CONTACTS	3094	3636	2212
RECALL@1 (%)	40.8	62.7	74.0
PERCENTAGE OF CONTACTS RESOLVED (%)	0.9	5.3	13.6

models on live customer contacts. We present the results of the model on delivery related issue.

5.2 Results

Table 5 shows the online results for the three models on live customer contacts: pretrained model (M) as explained in section 3, finetuned model (M_{BCE}) with BCE loss and data regularization; and finetuned model ($M_{ranking}$) with ranking loss and data regularization. For each model, we calculate Recall@1 as the fraction of total responses that were accepted by the agents. We also report the percentage of contacts resolved, which represents the percentage of contacts when the model's recommendations were accepted at every turn.

After finetuning using BCE, the Recall@1 of the model showed an absolute increase of 22% over the pretrained model. Finetuning using the ranking loss outperformed the pretrained model by 33%, and BCE finetuned model by 11%. For the ranking loss based finetuning, the percentage of contacts resolved completely using the model's recommendations went up by 13% compared to the pretrained model.

5.3 Error Analysis

To better understand the model's failure cases, we manually read 100 turns where the agents rejected the model's recommended response and typed on their own. For this study, we focus on the best model - finetuned using ranking loss and data regularization.

We found that 17% of the model's errors were caused because the conversation had gone off the common path. The model is not able to recommend the correct response in these cases because it has not seen these types of conversations during the training, leading to the template pool being unable to cover many of said cases. Examples of why conversations may go long or off-track include, but are not limited to, a customer being unhappy with the solution, experiencing multiple issues within the same chat, and/or participating in side conversations.

Another major reason for rejection was due to missing and/or incomplete context data available to the model. To contrast, this means agents had access to a richer profile information than what was available to the model. As a result, the model did not have the relevant context to recommend the right response. In 28% of the cases, there were extra profile features available to the agents, such as being able to check the carrier's website, that were not available to the model.

In dialogues, usually there is more than one correct response, giving room to agent's subjectivity in accepting/rejecting a model's response. We found that 15% of rejected turns occurred because the agent decided to reject the model's suggestion in favor of a stylistically different but semantically similar message. For example, some agents preferred closing the contact with 'Thank you for contacting 'while others preferred to directly say 'Take care and have a nice day '.

6 Conclusion and Future Work

In this paper, we study a less explored approach of finetuning a retrieval-based dialogue system based on a small amount of high-quality, annotated data that resulted in 26% offline and 33% online performance improvement in Recall@1 over a pretrained model.We deployed the model in an agent-support application, and demonstarte that the proposed model achieves 74% Recall@1, suggesting these models are effective in assisting agents by recommending text responses. The results demonstrate the effectiveness of pretrain-finetune approach in dealing with the limited supervised data challenge. In this paper, we focus on a customer service delivery issue, but since this technique can scale to other task-oriented dialog systems with a wide range of applications.

We believe that additional investment in contextual and profile features would help improve the model performance. As discussed in the error analysis section, 28% of the model's error is caused due to missing context information. In addition, manual study of the rejection reasons highlights the issue of subjective evaluation. More investment is needed in agent training and standardization of

annotation. We believe the model can significantly benefit from better annotation and evaluation.

References

Layla El Asri, Hannes Schulz, Shikhar Sharma, Jeremie Zumer, Justin Harris, Emery Fine, Rahul Mehrotra, and Kaheer Suleman. 2017. Frames: A corpus for adding memory to goal-oriented dialogue systems. *arXiv preprint arXiv:1704.00057*.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Inigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. Multiwoz-a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *arXiv preprint arXiv:1810.00278*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Tianxing He, Jun Liu, Kyunghyun Cho, Myle Ott, Bing Liu, James Glass, and Fuchun Peng. 2019. Mix-review: Alleviate forgetting in the pretrain-finetune framework for neural language generation models. *arXiv preprint arXiv:1910.07117*.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*.

Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young, et al. 2016. Smart reply: Automated response suggestion for email. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 955–964.

Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. 2015. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *arXiv preprint arXiv:1506.08909*.

Yichao Lu, Phillip Keung, Shaonan Zhang, Jason Sun, and Vikas Bhardwaj. 2017. A practical approach to dialogue response generation in closed domains. *arXiv preprint arXiv:1703.09439*.

Yichao Lu, Manisha Srivastava, Jared Kramer, Heba Elfardy, Andrea Kahn, Song Wang, and Vikas Bhardwaj. 2019. Goal-oriented end-to-end conversational models with profile features in a real-world setting. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 48–55.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.