ParlAI: A Dialog Research Software Platform

Alexander H. Miller, Will Feng, Adam Fisch, Jiasen Lu,
Dhruv Batra, Antoine Bordes, Devi Parikh and **Jason Weston**
Facebook AI Research

Abstract

We introduce ParlAI (pronounced "parlay"), an open-source software platform for dialog research implemented in Python, available at `http://parl.ai`. Its goal is to provide a unified framework for sharing, training and testing dialog models; integration of Amazon Mechanical Turk for data collection, human evaluation, and online/reinforcement learning; and a repository of machine learning models for comparing with others' models, and improving upon existing architectures. Over 20 tasks are supported in the first release, including popular datasets such as SQuAD, bAbI tasks, MCTest, WikiQA, QACNN, QADailyMail, CBT, bAbI Dialog, Ubuntu, OpenSubtitles and VQA. Several models are integrated, including neural models such as memory networks, seq2seq and attentive LSTMs.

1 Introduction

The purpose of language is to accomplish communication goals, which typically involve a dialog between two or more communicators (Crystal, 2004). Hence, trying to solve dialog is a fundamental goal for researchers in the NLP community. From a machine learning perspective, building a learning agent capable of dialog is also fundamental for various reasons, chiefly that the solution involves achieving most of the subgoals of the field, and in many cases those subtasks are directly impactful to the task.

On the one hand dialog can be seen as a single task (learning how to talk) and on the other hand as thousands of related tasks that require different skills, all using the same input and output format. The task of booking a restaurant, chatting about sports or the news, or answering factual or perceptually-grounded questions all fall under dialog. Hence, methods that perform task transfer appear useful for the end-goal. Memory, logical and commonsense reasoning, planning, learning from interaction, learning compositionality and other AI subgoals also have clear roles in dialog.

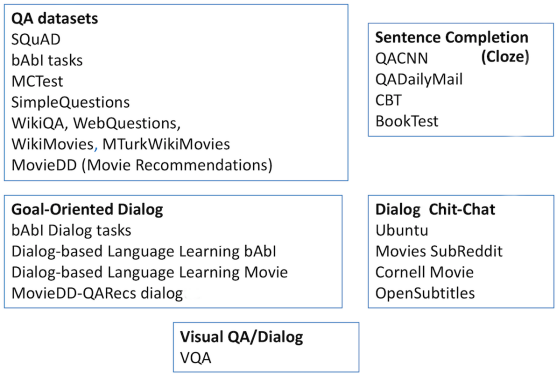

Figure 1: The tasks in the first release of ParlAI.

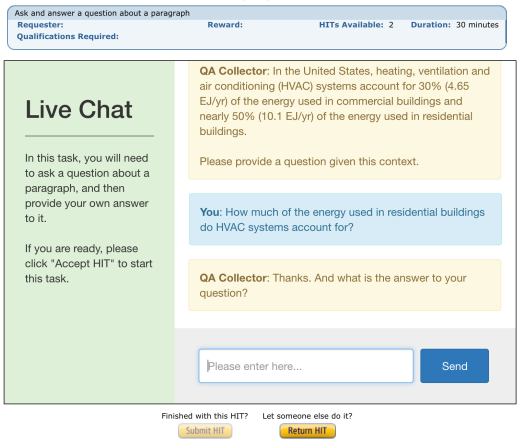

Figure 2: MTurk Live Chat for collecting QA datasets in ParlAI.

However, to pursue these research goals, software tools should unify the different dialog subtasks and the agents that can learn from them. Working on individual datasets can lead to siloed

research, where the overfitting to specific qualities of a dataset do not generalize to solving other tasks. For example, methods that do not generalize beyond WebQuestions (Berant et al., 2013) because they specialize on knowledge bases only, SQuAD (Rajpurkar et al., 2016) because they predict start and end context indices (see Sec. 7), or bAbI (Weston et al., 2015) because they use supporting facts or make use of its simulated nature.

In this paper we present a software platform, ParlAI (pronounced "par-lay"), that provides researchers a unified framework for training and testing dialog models, especially multitask training or evaluation over many tasks at once, as well as seamless integration with Amazon Mechanical Turk. Over 20 tasks are supported in the first release, including many popular datasets, see Fig. 1. Included are examples of training neural models with PyTorch and Lua Torch[1]. Using Theano[2] or Tensorflow[3] instead is also straightforward.

The overarching goal of ParlAI is to build a community-based platform for easy access to both tasks and learning algorithms that perform well on them, in order to push the field forward. This paper describes our goals in detail, and gives a technical overview of the platform.

2 Goals

The goals of ParlAI are as follows:

A unified framework for development of dialog models. ParlAI aims to unify dialog dataset input formats fed to machine learning agents to a *single format*, and to standardize evaluation frameworks and metrics as much as possible. Researchers can submit their new tasks and their agent training code to the repository to share with others in order to aid reproducibility, and to better enable follow-on research.

General dialog involving many different skills. ParlAI contains a seamless combination of real and simulated language datasets, and encourages multitask model development & evaluation by making multitask models as easy to build as single task ones. This should reduce overfitting of model design to specific datasets and encourage models that perform task transfer, an important prerequisite for a general dialog agent.

Real dialog with people. ParlAI allows collecting, training and evaluating on live dialog with humans via Amazon Mechanical Turk by making it easy to connect Turkers with a dialog agent, see Fig. 2. This also enables comparison of Turk experiments across different research groups, which has been historically difficult.

Towards a common general dialog model. Our aim is to motivate the building of new tasks and agents that move the field towards a working dialog model. Hence, each new task that goes into the repository should build towards that common goal, rather than being seen solely as a piece of independent research.

3 General Properties of ParlAI

ParlAI consists of a number of tasks and agents that can be used to solve them. All the tasks in ParlAI have a single format (API) which makes applying any agent to any task, or multiple tasks at once, simple. The tasks include both fixed supervised/imitation learning datasets (i.e. conversation logs) and interactive (online or reinforcement learning) tasks, as well as both real language and simulated tasks, which can all be seamlessly trained on. ParlAI also supports other media, e.g. images as well as text for visual question answering (Antol et al., 2015) or visually grounded dialog (Das et al., 2017). ParlAI automatically downloads tasks and datasets the first time they are used. One or more Mechanical Turkers can be embedded inside an environment (task) to collect data, train or evaluate learning agents.

Examples are included in the first release of training with PyTorch and Lua Torch. ParlAI uses ZeroMQ to talk to languages other than Python (such as Lua Torch). Both batch training and hogwild training of models are supported and built into the code. An example main for training an agent is given in Fig. 3.

4 Worlds, Agents and Teachers

The main concepts (classes) in ParlAI are worlds, agents and teachers:

- **world** – the environment. This can vary from being very simple, e.g. just two agents conversing, to much more complex, e.g. multiple agents in an interactive environment.
- **agent** – an agent that can act (especially, speak) in the world. An agent is either a learner (i.e. a machine learned system), a

[1] http://pytorch.org/ and http://torch.ch/
[2] http://deeplearning.net/software/theano/
[3] https://www.tensorflow.org/

```
teacher = SquadTeacher(opt)
agent = MyAgent(opt)
world = World(opt, [teacher, agent])
for i in range(num_exs):
  world.parley()
  print(world.display())
```

```
def parley(self):
  for agent in self.agents:
    act = agent.act()
    for other_agent in self.agents:
      if other_agent != agent:
        other_agent.observe(act)
```

Figure 3: ParlAI main for displaying data (top) and the code for the world.parley call (bottom).

```
python examples/display_data.py -t babi

[babi:Task1k:4]: The office is north of the kitchen.
The bathroom is north of the office.
What is north of the kitchen?
[cands: office|garden|hallway|bedroom|kitchen|bathroom]
   [RepeatLabelAgent]: office
- - - - - - - - - - - - - - - - - - - - - -
~~
[babi:Task1k:2]: Daniel went to the kitchen.
Daniel grabbed the football there.
Mary took the milk there.
Mary journeyed to the office.
Where is the milk?
[cands: office|garden|hallway|bedroom|kitchen|bathroom]
   [RepeatLabelAgent]: office
```

Figure 4: Example output to display data of a given task (see Fig. 3 for corresponding code).

hard-coded bot such as one designed to interact with learners, or a human (e.g. a Turker).

- **teacher** – a type of agent that talks to the learner in order to teach it, e.g. implements one of the tasks in Fig. 1.

After defining a world and the agents in it, a main loop can be run for training, testing or displaying, which calls the function world.parley() to run one time step of the world. Example code to display data is given in Fig. 3, and the output of running it is in Fig. 4.

5 Actions and Observations

All agents (including teachers) speak to each other in a single common format – the observation/action object (a python dict), see Fig. 5. It is used to pass text, labels and rewards between agents. The same object type is used for both talking (acting) and listening (observing), but with different values in the fields. Hence, the object is returned from agent.act() and passed in to agent.observe(), see Fig. 3.

The fields of the message are as follows:

- *text:* a speech act.
- *id:* the speaker's identity.
- *reward:* a real-valued reward assigned to the receiver of the message.
- *episode_done:* indicating the end of a dialog.

For supervised datasets, there are some additional fields that can be used:

- *label:* a set of answers the speaker is expecting to receive in reply, e.g. for QA datasets the right answers to a question.
- *label_candidates:* a set of possible ways to respond supplied by a teacher, e.g. for multiple choice datasets or ranking tasks.
- *text_candidates:* ranked candidate predictions from a learner. Used to evaluate ranking metrics, rather than just evaluate the single response in the *text* field.
- *metrics:* A teacher can communicate to a learning agent metrics on its performance.

Finally other media can also be supported with additional fields:

- *image:* an image, e.g. for Visual Question Answering or Visual Dialog datasets.

As the dict is extensible, we can add more fields over time, e.g. for audio and other sensory data, as well as actions other than speech acts.

Each of these fields are technically optional, depending on the dataset, though the *text* field will most likely be used in nearly all exchanges. A typical exchange from a ParlAI training set is shown in Fig. 6.

6 Code Structure

The ParlAI codebase has five main directories:

- **core**: the primary code for the platform.
- **agents**: contains agents which can interact with the worlds/tasks (e.g. learning models).
- **examples**: contains examples of different mains (display data, training and evaluation).
- **tasks**: contains code for the different tasks available from within ParlAI.
- **mturk**: contains code for setting up Mechanical Turk and sample MTurk tasks.

6.1 Core

The core library contains the following files:

- `agents.py`: defines the Agent base class for all agents, which implements the observe() and act() methods, the Teacher class which also reports metrics, and MultiTaskTeacher for multitask training.

Observation/action `dict`
Passed back and forth between agents & environment.

Contains:

.text	*text of speaker(s)*
.id	*id of speaker(s)*
.reward	*for reinforcement learning*
.episode_done	*signals end of episode*

For supervised dialog datasets:

.label	
.label_candidates	*multiple choice options*
.text_candidates	*ranked candidate responses*
.metrics	*evaluation metrics*

Other media:

.image	*for VQA or Visual Dialog*

Figure 5: The observation/action dict is the central message passing object in ParlAI: agents send this message to speak, and receive a message of this form to observe other speakers and the environment.

```
Teacher: {
    'text': 'Sam went to the kitchen.\n Pat gave Sam the
milk.\nWhere is the milk?',\\
    'labels': ['kitchen'],
    'label_candidates': ['hallway', 'kitchen', 'bathroom'],
    'episode_done': False
}

Student: {
   'text': 'hallway'
}

Teacher: {
    'text': 'Sam went to the hallway\nPat went to the
bathroom\nWhere is the milk?',
    'labels': ['hallway'],
    'label_candidates': ['hallway', 'kitchen', 'bathroom'],
    'done': True
}

Student: {
    'text': 'hallway'
}
...
```

Figure 6: A typical exchange from a ParlAI training set involves messages passed using the observation/action dict (the test set would not include labels). Shown here is the bAbI dataset.

- `dialog_teacher.py`: the base teacher class for doing dialog with fixed chat logs.
- `worlds.py`: defines the base World class, DialogPartnerWorld for two speakers, MultiAgentDialogWorld for more than two, and two containers that can wrap a chosen environment: BatchWorld for batch training, and HogwildWorld for training across multiple threads.
- `dict.py`: code for building language dictionaries.
- `metrics.py`: computes exact match, F1 and ranking metrics for evaluation.
- `params.py`: uses argparse to interpret command line arguments for ParlAI

6.2 Agents

The agents directory contains machine learning agents. Currently available within this directory:

- **drqa:** an attentive LSTM model DrQA (Chen et al., 2017) implemented in PyTorch that has competitive results on SQuAD (Rajpurkar et al., 2016) amongst other datasets.
- **memnn:** code for an end-to-end memory network (Sukhbaatar et al., 2015) in Lua Torch.
- **remote_agent:** basic class for any agent connecting over ZeroMQ.
- **seq2seq:** basic GRU sequence to sequence model (Sutskever et al., 2014)
- **ir_baseline:** information retrieval (IR) baseline that scores responses with TFIDF-weighted matching (Ritter et al., 2011).
- **repeat_label:** basic class for merely repeating all data sent to it (e.g. for debugging).

6.3 Examples

This directory contains examples of different mains:.

- `display_data`: display data from a particular task provided on the command-line.
- `display_model`: show the predictions of a provided model.
- `eval_model`: compute evaluation metrics for a given model on a given task.
- `train_model`: execute a standard training procedure with a given task and model, including logging and possibly alternating between training and validation.

For example, one can display 10 random examples from the bAbI tasks (Weston et al., 2015):

```
python display_data.py -t babi -n 10
```

Display multitasking bAbI and SQuAD (Rajpurkar et al., 2016) at the same time:

```
python display_data.py -t babi,squad
```

Evaluate an IR baseline model on the Movies Subreddit:

```
python eval_model.py -m ir_baseline -t
'#moviedd-reddit' -dt valid
```

Train an attentive LSTM model on the SQuAD dataset with a batch size of 32 examples:

```
python train_model.py -m drqa -t squad
-b 32
```

6.4 Tasks

Over 20 tasks are supported in the first release, including popular datasets such as SQuAD (Rajpurkar et al., 2016), bAbI tasks (Weston et al., 2015), QACNN and QADailyMail (Hermann et al., 2015), CBT (Hill et al., 2015), bAbI Dialog tasks (Bordes and Weston, 2016), Ubuntu (Lowe et al., 2015) and VQA (Antol et al., 2015). All the datasets in the first release are shown in Fig. 1[4].

The tasks are separated into five categories:

- Question answering (QA): one of the simplest forms of dialog, with only 1 turn per speaker. Any intelligent dialog agent should be capable of answering questions, and there are many kinds of questions (and hence datasets) that one can build, providing a set of very important tests. Question answering is particularly useful in that the evaluation is simpler than other forms of dialog if the dataset is labeled with QA pairs and the questions are mostly unambiguous.
- Sentence Completion (Cloze Tests): the agent has to fill in a missing word in the next utterance in a dialog. Again, this is specialized dialog task, but it has the advantage that the datasets are cheap to make and evaluation is simple, which is why the community has built several such datasets.
- Goal-Oriented Dialog: a more realistic class of tasks is where there is a goal to be achieved by the end of the dialog. For example, a customer and a travel agent discussing a flight, one speaker recommending another a movie to watch, and so on.
- Chit-Chat: dialog tasks where there may not be an explicit goal, but more of a discussion — for example two speakers discussing sports, movies or a mutual interest.
- Visual Dialog: dialog is often grounded in physical objects in the world, so we also include dialog tasks with images as well as text.

Choosing a task in ParlAI is as easy as specifying it on the command line, as shown in the dataset display utility, Fig. 4. If the dataset has not been used before, ParlAI will automatically download it. As all datasets are treated in the same way in ParlAI (with a single dialog API, see Sec. 5), a dialog agent can switch training and testing between any of them. Importantly, one can specify many tasks at once (multitasking) by simply providing a comma-separated list, e.g. the command line arguments `-t babi,squad`, to use those two datasets, or even all the QA datasets at once (`-t #qa`) or indeed every task in ParlAI at once (`-t #all`). The aim is to make it easy to build and evaluate very rich dialog models.

Each task is contained in a folder with the following standardized files:

- `build.py`: file for setting up data for the task, including downloading the data the first time it is requested.
- `agents.py`: contains teacher class(es), agents that live in the world of the task.
- `worlds.py`: optionally added for tasks that need to define new/complex environments.

To add a new task, one must implement build.py to download any required data, and agents.py for the teacher. If the data consist of fixed logs/dialog scripts such as in many supervised datasets (SQuAD, Ubuntu, etc.) there is very little code to write. For more complex setups where an environment with interaction has to be defined, new worlds and/or teachers can be implemented.

6.5 Mechanical Turk

An important part of ParlAI is seamless integration with Mechanical Turk for data collection, training or evaluation. Human Turkers are also viewed as agents in ParlAI and hence human-human, human-bot, or multiple humans and bots in group chat can all converse within the standard framework, switching out the roles as desired with no code changes to the agents. This is because Turkers also receive and send via the same interface: using the fields of the observation/action dict. We provide two examples in the first release:

(i) **qa_collector:** an agent that talks to Turkers to collect question-answer pairs given a context paragraph to build a QA dataset, see Fig. 2.

(ii) **model_evaluator:** an agent which collects ratings from Turkers on the performance of a bot on a given task.

Running a new MTurk task involves implementing and running a main file (like run.py) and defining several task specific parameters for the world and agent(s) you wish humans to talk to. For data collection tasks the agent should pose the problem and ask the Turker for e.g. the answers to questions, see Fig. 2. Other parameters include the task description, the role of the Turker in the

[4] All dataset descriptions and references are at `http://parl.ai` in the `README.md` and `task_list.py`.

task, keywords to describe the task, the number of hits and the rewards for the Turkers. One can run in a sandbox mode before launching the real task where Turkers are paid.

For online training or evaluation, the Turker can talk to your machine learning agent, e.g. LSTM, memory network or other implemented technique. New tasks can be checked into the repository so researchers can share data collection and data evaluation procedures and reproduce experiments.

7 Demonstrative Experiment

To demonstrate ParlAI in action, we give results in Table 1 of DrQA, an attentive LSTM architecture with single task and multitask training on the SQuAD and bAbI tasks, a combination not shown before with any method, to our knowledge.

This experiment simultaneously shows the power of ParlAI — how easy it is to set up this experiment — and the limitations of current methods. Almost all methods working well on SQuAD have been designed to predict a phrase from the given context (they are given labeled start and end indices in training). Hence, those models cannot be applied to all dialog datasets, e.g. some of the bAbI tasks include yes/no questions, where yes and no do not appear in the context. This highlights that researchers should not focus models on a single dataset. ParlAI does not provide start and end label indices as its API is dialog only, see Fig. 5. This is a deliberate choice that discourages such dataset overfitting/ specialization. However, this also results in a slight drop in performance because less information is given[5] (66.4 EM vs. 69.5 EM, see (Chen et al., 2017), which is still in the range of many existing well-performing methods, see `https://stanford-qa.com`).

Overall, while DrQA can solve *some* of the bAbI tasks and performs well on SQuAD, it does not match the best performing methods on bAbI (Seo et al., 2016; Henaff et al., 2016), and multitasking does not help. Hence, ParlAI lays out the challenge to the community to find learning algorithms that are generally applicable and that benefit from training over many dialog datasets.

[5] As we now do not know the location of the true answer, we randomly pick the start and end indices of any context phrase matching the given training set answer, in some cases this is unique.

	Task	Single	Multitask
bAbI 10k	1: Single Supporting Fact	100	100
	2: Two Supporting Facts	98.1	54.3
	3: Three Supporting Facts	45.4	58.1
	4: Two Arg. Relations	100	100
	5: Three Arg. Relations	98.9	98.2
	11: Basic Coreference	100	100
	12: Conjunction	100	100
	13: Compound Coref.	100	100
	14: Time Reasoning	99.8	99.9
	16: Basic Induction	47.7	48.2
SQuAD	(Dev. Set)	66.4	63.4

Table 1: Test Accuracy of DrQA on bAbI 10k and SQuAD (Exact Match metric) using ParlAI. The subset of bAbI tasks for which the answer is exactly contained in the text is used.

8 Related Software

There are many existing independent dialog datasets, and training code for individual models that work on some of them. Many are framed in slightly different ways (different formats, with different types of supervision), and ParlAI attempts to unify this fragmented landscape.

There are some existing software platforms that are related in their scope, but not in their specialization. OpenAI's Gym and Universe[6] are toolkits for developing and comparing reinforcement learning (RL) algorithms. Gym is for games like Pong or Go, and Universe is for online games and websites. Neither focuses on dialog or covers the case of supervised datasets as we do.

CommAI[7] is a framework that uses textual communication for the goal of developing artificial general intelligence through incremental tasks that test increasingly more complex skills, as described in (Mikolov et al., 2015). CommAI is in a RL setting, and contains only synthetic datasets, rather than real natural language datasets as we do here. In that regard it has a different focus to ParlAI, which emphasizes the more immediate task of real dialog, rather than directly on evaluation of machine intelligence.

9 Conclusion and Outlook

ParlAI is a framework allowing the research community to share existing and new tasks for dialog as well as agents that learn on them, and to collect and evaluate conversations between agents and humans via Mechanical Turk. We hope this

[6] `https://gym.openai.com/` and `https://universe.openai.com/`

[7] `https://github.com/facebookresearch/CommAI-env`

tool enables systematic development and evaluation of dialog agents, helps push the state of the art in dialog further, and benefits the field as a whole.

Acknowledgments

We thank Mike Lewis, Denis Yarats, Douwe Kiela, Michael Auli, Y-Lan Boureau, Arthur Szlam, Marc'Aurelio Ranzato, Yuandong Tian, Maximilian Nickel, Martin Raison, Myle Ott, Marco Baroni, Leon Bottou and other members of the FAIR team for discussions helpful to building ParlAI.

References

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. VQA: Visual Question Answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2425–2433.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, volume 2, page 6.

Antoine Bordes and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. *arXiv:1704.00051*.

David Crystal. 2004. *The Cambridge encyclopedia of the English language*. Ernst Klett Sprachen.

Abhishek Das, Satwik Kottur, José MF Moura, Stefan Lee, and Dhruv Batra. 2017. Learning cooperative visual dialog agents with deep reinforcement learning. *arXiv preprint arXiv:1703.06585*.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2016. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2015. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301*.

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*.

Tomas Mikolov, Armand Joulin, and Marco Baroni. 2015. A roadmap towards machine intelligence. *arXiv preprint arXiv:1511.08130*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv:1606.05250*.

Alan Ritter, Colin Cherry, and William B Dolan. 2011. Data-driven response generation in social media. In *EMNLP*, pages 583–593. Association for Computational Linguistics.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Query-reduction networks for question answering. *arXiv preprint arXiv:1606.04582*.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

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:1502.05698*.