General AI Challenge
Specifications of the First (Warm-Up) Round: Gradual Learning - Learning Like a Human

Please read me!

Marek Rosa, GoodAI, 13.3.2017
marek.rosa@goodai.com
www.GoodAI.com
www.general-ai-challenge.org

Version 2

Table of Contents

Introduction

This is the first (warm-up) round of the General AI Challenge. It won't be the last.

We must assume that even the winning implementations may have limitations, errors, missing features, or that we will discover that we need slightly different requirements in order to get us closer to general AI. We expect to learn from this round a lot and reflect the learned lessons in the next round.

The environment used in this first round is very constrained, simple, and relatively easy to predict.
That's because in this round, we wanted to isolate the gradual learning ability and focus solely on that. We plan to use much more complex, stochastic environments in future rounds.

We would like to emphasize that this round and the tasks are not aiming for development of an NLP agent: instead, we are aiming for an agent with more universal skills. The agent should exhibit gradual learning capabilities, irrespective of the format of the input data. We have chosen language-based tasks in this round purely out of convenience, since the design of gradual tasks in such an environment is much more intuitive to us.

About

The purpose of this document is to define the Gradual Learning round, tasks, interfaces between an agent and the environment, evaluation process, etc. Please read this document to make sure your next steps are clear.

Objective

The objective is to build an agent that acquires skills in a gradual manner and uses existing skills to learn and solve novel tasks more efficiently, needing less training examples.

This round is not concerned with how good an agent is at solving a particular task (e.g. highest score in a game).

Goals of the Round

- Scientific:
    - To get working examples of agents that can acquire skills in a gradual manner and use learned skills to learn new skills (increasing the efficiency of learning).
    - We are not optimizing for agent's performance of existing skills (how good an agent is at delivering solutions for problems it knows). Instead, we are optimizing for agent's performance on solving new/unseen problems.
    - Example:
        - if an agent is presented with a new/unseen problem, how fast (i.e. in how many simulation steps) will it deliver a good solution? This also includes a question of how fast the agent will be

at discovering this new solution. If the agent has already learned to find solutions for similar problems, it should use existing skills in order to discover the new skill.
    - Agents must provably use gradual learning and will be evaluated on how fast they are (how many simulation steps do they need) at discovering acceptable solutions to new tasks.
    - Agents won't be evaluated on task performance, cumulative reward etc.
- In the context of the overall General AI Challenge:
    - To kick off the Challenge series.
    - For us organizers, to get experience in challenge design.
    - To learn and prepare to iterate with the next Round.

Prizes

1. Quantitative prize: for the agent that solves all evaluation tasks in the shortest number of simulation steps through gradual learning
    a. This prize will use objective evaluation / black-box testing (no jury, no evaluation of designs, etc)
    1st place: $15,000
    2nd place: $10,000
    3rd place: $5,000

    Total: $30,000

2. Qualitative prize: for the idea, concept, or design that shows the best promise for scalable gradual learning (this does not have to be a working AI agent; an idea described in a white paper alone can also win the prize)
    a. This prize will be subjective and evaluated by the jury through white-box evaluation
    b. The jury white-box evaluates the top 10 submissions from the quantitative prize along with "wild-cards" (at least 10 submissions pre-selected by GoodAI team)
    c. We do not guarantee feedback on the submitted solutions
    1st place: $10,000
    2nd place: $7,000
    3rd place: $3,000

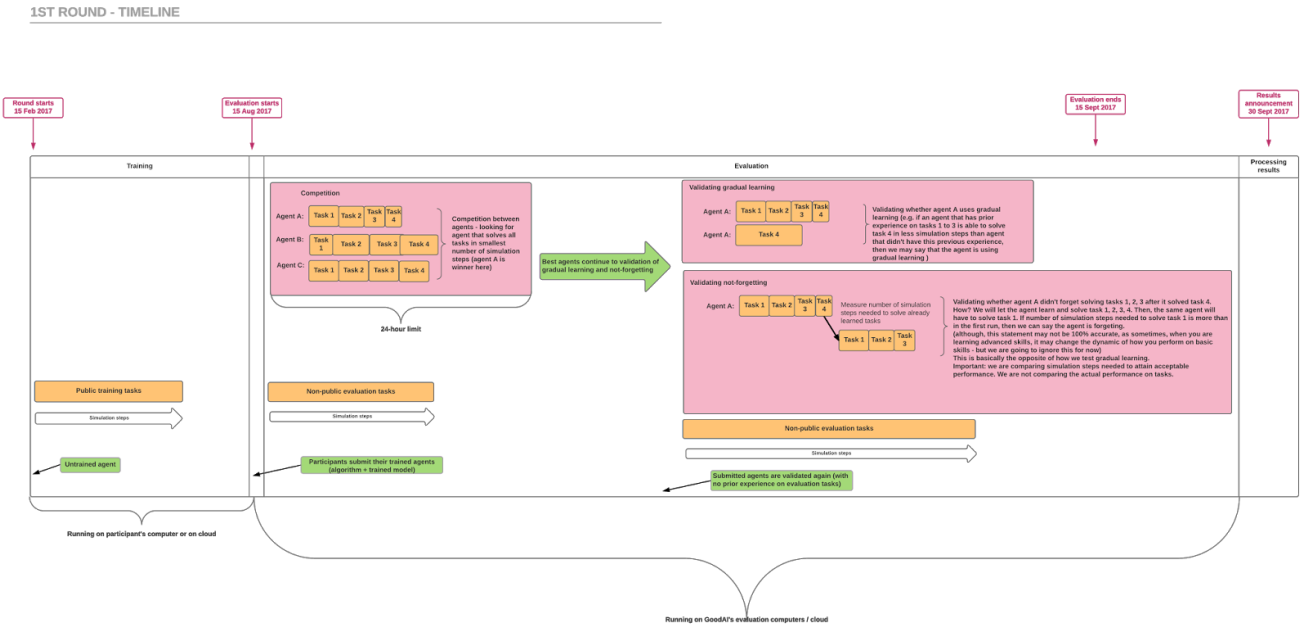
    Total: $20,000

In addition to the monetary prize, you will get a chance to win an NVIDIA GPU.
All winners will receive a printed and signed diploma.

Timeline of the Round & What do You need to Submit at the End

- Round announcement & start: **15 February 2017**
- Evaluation start: **15 August 2017 (+6 months)**
    - Solution submissions are **due by August 14th 2017, 23:59 CET (code and whitepaper describing your solution)** to solved@general-ai-challenge.org
    - You can send as many solutions as you'd like, but each must be significantly different from others

- Evaluation end: **15 September 2017 (+1 month)**
- Results announcement: **30 September 2017 (+2 weeks)**



Full resolution diagram available at:
https://drive.google.com/file/d/0B820uHFOHp0mODQ3SFpzV01JTnc/view?usp=sharing

- **Training phase**
  - Participants develop and train their agents
  - Participants can pre-train their agents on our training tasks or they can create their own training tasks.
  - Prior knowledge (pre-training) is ok
  - To train their agents, participants can use free Azure cloud space provided by our technological partner Microsoft
  - Participants' agents will be evaluated on tasks that are not part of the training set (participants are encouraged to not overfit their agents on the training tasks, as these only remotely resemble the nonpublic evaluation tasks)
  - Training tasks will serve as a "demo" for the real evaluation phase – people can benchmark their agents on the tasks and see results in order to get a taste of how their agents might work in the evaluation phase
- **Evaluation phase**
  - **Contestants will send an email to solved@general-ai-challenge.org, containing the following:**
    - the source code of their agent (in any programming language),
    - the training tasks (and training data) used for training the agent (only in case they are different from the training tasks that the organizers provided)
    - their pre-trained agent,
    - the design (description/explanation, white paper) of their agent. This white paper should:

        - be brief and well-structured (2 pages max.; in case the whitepaper exceeds the 2-page limit, participants must include a one-page summary of the paper at the beginning),
        - include instructions on how to run the agent, including if it should be evaluated on GPU or CPU,
        - explain the main principles and motivations behind the agent's design in a brief, structured manner,
        - include participant's / team's name and contact details,
        - state their preference, whether they want to open source the agent, or just share it with the organizers
      - note: if the contestant decides to compete for the qualitative prize only (best idea), he or she can submit the white paper only.
  - It must be possible for us to reproduce the agent based on the submitted source code and training data.
  - Organizers will run submitted pre-trained agents on nonpublic evaluation tasks.
  - Every agent will have 24 hours to solve all evaluation tasks. The agent that solves the evaluation tasks in the smallest number of simulation steps will be evaluated for presence of gradual learning and not-forgetting afterwards (might be performed on multiple computers in parallel). If this agent does not pass the gradual learning and not-forgetting evaluation, second runner-up will be evaluated and so on.
  - The evaluation environment and the agent will run on the same computer, managed by organizers.
- **Results**
  - We will make evaluation tasks publicly available after the announcement of the round winners (evaluation tasks will not be public until this point).

Evaluation criteria

(A reminder what is the goal this Round:

- *to get working examples of agents that can acquire skills in a gradual manner and use learned skills to learn new skills (increasing the efficiency of learning). Agents must provably use gradual learning.)*

Quantitative prize:

The evaluation timeline for the Quantitative prize is demonstrated on the timeline diagram above (accessible here in full resolution: https://drive.google.com/file/d/0B820uHFOHp0mODQ3SFpzV01JTnc/view?usp=sharing)

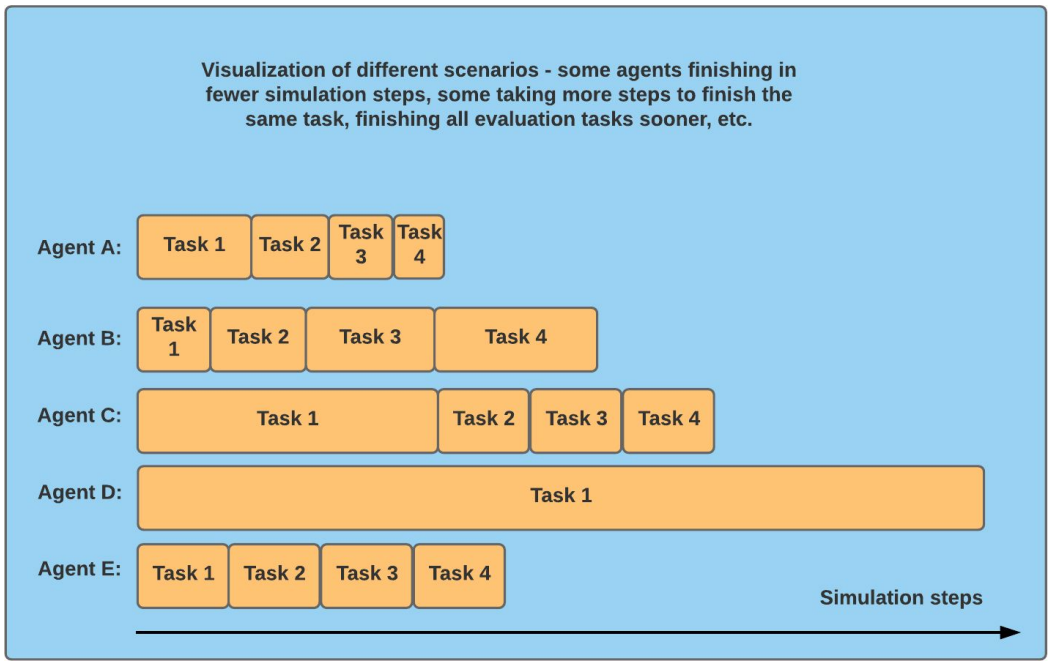
1. The agent that solves all evaluation tasks in the **shortest number of simulation steps** (c.f. section A.1.2) will win, if it provably demonstrates gradual learning and not-forgetting (see below). The evaluation tasks will be different from public training tasks. A suggestion on what the evaluation tasks might look like can be seen in Appendix A, section A.1.6.

   Every submitted agent will have 24 hours to solve all evaluation tasks. The agent should be able to solve more than 1 task instance; a high number of successfully solved instances in a row indicate that it's not pure luck. The agent that solves the evaluation tasks in the smallest number of simulation steps will be evaluated for presence of gradual learning and not-forgetting afterwards (might be performed on multiple computers in parallel) - see below. If this agent does not pass the gradual learning and not-forgetting evaluation, second

runner-up will be evaluated and so on.

2. The agent must demonstrate **gradual learning** – the ability to use previously learned skills to learn new skills faster
    a. An example of how to validate whether an agent uses gradual learning:
        i. The submitted and pre-trained agent runs on tasks 1, 2, 3 and 4.
        ii. The same agent (without experience on tasks 1, 2, 3) is then executed directly on task 4. This agent has skipped tasks 1, 2, 3; its only experience is task 4.
        iii. If an agent that had experience on tasks 1, 2, 3 solves task 4 in less simulation steps than agent that didn't have experience on tasks 1, 2, 3, then we assume that the agent might be using gradual learning
        iv. We are aware that the above property does not guarantee for 100% that the agent used gradual learning, but we have decided to go with the our best idea so far how to validate it.
            1. We will define tasks 1, 2, 3, 4 to have "conceptual" dependencies, so we can be almost sure that 4 requires 1, 2, 3, and 3 requires 1, 2, etc.
        v. Gradual learning will be validated on multiple tasks and in various orders.
3. Agents must not forget skills
    a. For example: after solving tasks 1, 2, 3 and then 4, agent must still be able to solve tasks 1,2,3 (with the same or better performance)
4. An agent has to solve a task only up to **"acceptable performance"**. Achieving maximum performance (score, reward) is not required. Immediately after the agent reaches acceptable performance, the environment presents it the next task. This way, the only thing that matters is whether the agent has solved a task. Its actual performance on that task is not relevant. This leads to focusing on "efficiency of learning" instead of "performance on tasks".

Example:



Full resolution diagram available at:
https://drive.google.com/file/d/0B820uHFOHp0mNGt4U3czVXJ2RWM/view?usp=sharing

Qualitative prize:

The Qualitative prize is awarded for the idea, concept, or design that shows the best promise for scalable gradual learning. The submission does not have to be a working AI agent; an idea described in a white paper alone can also win the prize, although a working AI agent solution will have better chances. This prize is subjective and will be evaluated by the jury through white-box evaluation. The jury will evaluate the top 10 submissions from the quantitative prize along with "wild-cards" (at least 10 submissions pre-selected by GoodAI team).

Rules

1. **The solution must be able to use finite computational resources** (it should be able to pass evaluation tasks in 24 hours; overall complexity of evaluation task set is comparable to complexity of training task set). The solutions will be bound by configuration of the evaluation hardware, which will be roughly equivalent to:
    1. GPU-bound solutions: Microsoft Azure NV6 virtual machine (6 cores, 56 GB RAM, 340 GB disk, NVIDIA M60 GPU)
    2. CPU-bound solutions: Microsoft Azure F16 virtual machine (Intel Xeon E5-2673 v3 with 16 cores, 32 GB RAM, 256 GB disk, no GPU)

    If a participant's' solution is GPU-bound, it will be evaluated on the first hardware configuration. CPU-bound solutions will be evaluated on the second hardware configuration. The participants will choose which hardware configuration should be used for evaluating their solution.
2. Tasks are **ordered** (training and evaluation), because we need to help agents to learn skills that can be then used to solve subsequent tasks.
3. Participants can form or join any number of teams
4. Teams can submit any number of submissions, but each submission must be significantly different (for example, just changing a constant in first submission isn't sufficient for warranting a second submission)
5. Participants are free to use any resources, e.g. open source libraries etc., as long as it won't prevent the organizers from running the participant's' code in the evaluation phase or building upon this code in the organizers' further research and business.
6. For teams or individuals who registered to participate and who would like to use Azure Cloud, Challenge partner Microsoft Czech Republic and Slovakia will open a BizSpark account that will include 5 MSDN subscriptions with a free credit of 150$ per subscription per month. The monthly free credit offer is valid for one year. The participants will be able to use the free credit for renting any virtual machine on Azure and running their solution there. In order to use the Azure offer, participants need to contact Tomas Prokop (b-toprok@microsoft.com) and attach a copy of the registration confirmation e-mail.

Judges

Selected GoodAI team members and the Challenge Advisory Board. GoodAI's CTO Marek Rosa and the team will have the final call in any judgements.

Environment & Agent Interface

- Agent's input:
  - one byte each simulation step
  - the environment and the agent are synced (one waits for the other).
- Agent's output:
  - one byte each simulation step
- Error signal:
  - one number in the range {-1,0,1} per one simulation step
  - correct actions will be communicated with the value +1
  - wrong actions will be communicated with the value -1
  - indifferent actions will be communicated with the value 0
- The following pseudo-code describes the way tasks are presented to the agent.
- *For each task T in a sequence of tasks, loop:*
  - ***while** T not solved:*
    - *Generate new instance I(T) of task T.*
    - ***while** I(T) is not solved, loop:*
      - *The environment computes the next state of I(T). If I(T) is solved or failed, **break**.*
      - *The agent receives the input and error signal for its previous action*
      - *Agent processes the input*
      - *Agent sends output to environment*
    - *if enough instances of T were solved, declare T as solved and **break**. (the actual rule for declaring T as solved may be more complex)*
    - ***continue** to next task instance I(T)*
  - ***continue** to next task T*
- There's no distinction between training and testing; agent can learn and respond at every step
- Agent doesn't have any special information about what task it is in – for agents, the environment is a continuous stream of input and output messages
- The environment will show the next task to the agent only after it solved the previous task with an acceptable performance.

More detailed description of the above is available in Appendix A.

Training Tasks

The training tasks fall into two groups:

1. Simple, introductory training tasks, called "micro-tasks", described in Appendix A of this document,
2. More advanced training tasks, called "mini-tasks", directly based on the CommAI-mini set.

The challenge uses a slightly modified version of CommAI-Env as its training and evaluation environment. Beside the modifications, It includes the implementations of the micro-tasks and of the mini-tasks.

Your Next Steps

1. Read Appendix A of this document to understand in detail the agent-environment interface, evaluation, and the micro-tasks.
2. Read the description of CommAI-mini set in CommAI: Evaluating the first steps towards a useful general AI.
3. Download the implementation of micro-tasks and of mini-tasks in the modified CommAI-env.

FAQ

- **Q:** What if several agents solve all tasks in similar number of simulation steps?
- **A:** Still the agent with smallest number of simulation steps will win.

- **Q:** Is this round going to have any restrictions related to open source software licenses? Are participants free to use AGPL codebases for the Challenge?
- **A:** No restrictions. You can use whatever you prefer, as long as the organizers will be able to run it on their servers during the evaluation phase, and use it freely in in their future work.

- Q: Is there a chance that the winning agents can scale up their capabilities to more noisy and complex environments?
- A: Maybe yes, maybe no. Nevertheless, we will learn valuable lessons on principles such as gradual learning, not forgetting, reuse of skills, etc -- even if they will be with respect to the text domain. We would like to emphasize that this round and the tasks are not aiming for development of an NLP agent: instead, we are aiming for an agent with universal skills. The agent should be able to learn to exhibit gradual learning capabilities, no matter the format of the input data. We have chosen language-based tasks in this round purely out of convenience, since the design of gradual tasks in such an environment is much more intuitive to us.

Reference materials

- Selected research publications:
    - A Roadmap towards Machine Intelligence, Tomas Mikolov, Armand Joulin, Marco Baroni, 2015/11. *An interesting proposal for a language-based environment for raising intelligent agents.*
    - A Survey on Transfer Learning, Sinno Jialin Pan, Qiang Yang, 2010/10. *This paper provides a systematic overview of approaches to transfer learning.*
    - CommAI: Evaluating the first steps towards a useful general AI, Marco Baroni, Armand Joulin, Allan Jabri, Germàn Kruszewski, Angeliki Lazaridou, Klemen Simonic, Tomas Mikolov, 2017/01. *Description of desiderata for a general AI, together with a description of an evaluation platform and a set of testing tasks.*

- Curriculum Learning, Yoshua Bengio, Jérôme Louradour, Ronan Collobert, Jason Weston, 2009/06. *This paper introduces* curriculum learning*: learning accelerated by presenting easy examples first and progressively increasing the difficulty. The paper was a product of the BabyAI project. See also Bengio's presentation on Deep Architectures for Baby AI.*
- Data-Efficient Learning of Feedback Policies from Image Pixels using Deep Dynamical Models, John-Alexander M. Assael, Niklas Wahlstrom, Thomas B. Schon, Marc Peter Deisenroth, 2015/10. *A network architecture composed of multiple parts, one for encoding the current state of the environment into more compressed representation. The other for predicting next state in this compressed state.*
- How the brain might work: A hierarchical and temporal model for learning and recognition (book), Dileep George, 2008/06. *Presents a theory about bottom-up and top-down processing of information, the importance of both of those directions, the role of spatial and temporal patterns and how they are connected to the neocortex.*
- Learning to compose neural networks for question answering, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, Dan Klein, 2016/01. *Learning how to compose (modular) neural networks to solve more complex tasks. One part of the algorithm learns how to the compose the modules and, in parallel, the specific modules are learned.*
- Learning without Forgetting, Zhizhong Li, Derek Hoiem, 2016/06. *The authors present a method for adding capabilities to convolutional neural networks (CNNs) while retaining existing capabilities.*
- Neural Programmer: Inducing Latent Programs with Gradient Descent, Arving Neelakantan, Quoc V. Lee, Ilya Sutskever, 2016/01. *This paper uses weak supervision and gradient descent to train an agent to perform table lookups on data using a number of intrinsic operators. The agent is trained only with what the answer should be, so it tries to converge onto a sequence of instructions which satisfy the answer. If the NPI paper is for the architecture, this is for the method.*
- Neural Programmer-Interpreters, Scott Reed, Nando de Freitas, 2016/02. *Winner of best paper award at ICLR 2016. This paper uses strong supervision to train a large controller which is able to compose hard coded instructions into programs. The methodology is very general as multiple instruction sets are shown on a diverse array of tasks. And the approach generalises to larger task variants quite well.*
- Never-Ending Learning, T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, J. Welling. 2015*. NELL is a semantic learning system that is continuously automatically expanding its knowledge base with the eventual goal of answering questions given by a human in natural language.*
- Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem, Jürgen Schmidhuber, 2011. *The author presents a framework for automatically discovering problems inspired by playful behavior in animals and humans.*
- Progressive Neural Networks, Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, Raia Hadsell, 2016/06. *Progressive neural networks adapt to new tasks by growing new columns while retaining previously acquired knowledge.*
- The Cascade-Correlation Learning Architecture, Scott Fahlman Christian, Christian Lebiere, 1990. *The Cascade-Correlation architecture accelerates learning by adding one hidden neuron at a time, keeping the preceding hidden weights frozen.*
- The Measure of All Minds: Evaluating Natural and Artificial Intelligence, José Hernández-Orallo, 2017. *The author compares the evaluation of intelligence in natural and artificial organisms, analyzes the*

usefulness of human-targeted tests for AI evaluation and proposes methods for evaluating AI based on algorithmic information theory.

- Other relevant research publications:
  - A Constrained Optimization Approach to Preserving Prior Knowledge During Incremental Training, S. Ferrari, M. Jensenius, in IEEE Transactions on Neural Networks (2008)
  - A Reinforcement Learning Architecture that Transfers Knowledge between Skills when Solving Multiple Tasks, Paolo Tommasino, Daniele Caligiore, Marco Mirolli, Gianluca Baldassarre, in IEEE Transactions on Cognitive and Developmental Systems (2016)
  - A survey on concept drift adaptation, João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, Abdelhamid Bouchachia, in ACM Computing Surveys (2014)
  - Active Long Term Memory Networks, Tommaso Furlanello, Jiaping Zhao, Andrew M. Saxe, Laurent Itti, Bosco S. Tjan, (2016)
  - Alleviating Catastrophic Forgetting via Multi-Objective Learning, Yaochu Jin, B. Sendhoff, in The 2006 IEEE International Joint Conference on Neural Network Proceedings (2006)
  - Catastophic Forgetting in Neural Networks, Ole-Marius Moe-Helgesen, Håvard Stranden, (2005)
  - ELLA: An efficient lifelong learning algorithm, Paul Ruvolo, Eric Eaton, in Proceedings of the 30th International Conference on Machine Learning (2013)
  - iCaRL: Incremental Classifier and Representation Learning, Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Christoph H. Lampert, (2016)
  - Learning and Transfer of Modulated Locomotor Controllers, Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, David Silver, (2016)
  - Learning Graphical State Transitions, Daniel D. Johnson, in ICLR (2017)
  - Improving Scalability of Reinforcement Learning by Separation of Concerns, Harm van Seijen, Mehdi Fatemi, Joshua Romoff, (2016)
  - Incremental learning with multi-level adaptation, Abdelhamid Bouchachia, in Neurocomputing (2011)
  - Learning to Learn for Global Optimization of Black Box Functions, Yutian Chen, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Timothy P. Lillicrap, Nando de Freitas (2016)
  - Learning to reinforcement learn, Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, et al. (2016)
  - Life-long learning Cell Structures—continuously learning without catastrophic interference, Fred H Hamker, in Neural Networks (2001)
  - Lifelong Learning with Non-i.i.d . Tasks, Anastasia Pentina, Christoph H Lampert, in NIPS (2015)
  - Lifelong Learning with Weighted Majority Votes, Anastasia Pentina, Ruth Urner, in NIPS (2016)
  - Lifelong Robot Learning, Sebastian Thrun, Tom M. Mitchell (1994)
  - Metacontrol for Adaptive Imagination-Based Optimization, Jessica B. Hamrick, Andrew J. Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, Peter W. Battaglia, in ICLR (2017)
  - Overcoming catastrophic forgetting in neural networks, James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, et al. (2016)
  - PathNet: Evolution Channels Gradient Descent in Super Neural Networks, Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, et al. (2017)

    - RL2: Fast Reinforcement Learning via Slow Reinforcement Learning, Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, Pieter Abbeel, in ICLR (2017)
    - Scalable Lifelong Learning with Active Task Selection, Paul Ruvolo, Eric Eaton, (2013)
    - The Forget-me-not Process, Kieran Milan, Joel Veness, James Kirkpatrick, Michael Bowling, Anna Koop, Demis Hassabis, in NIPS (2016)
    - Using Noise to Compute Error Surfaces in Connectionist Networks: A Novel Means of Reducing Catastrophic Forgetting, Robert M. French, Nick Chater, in Neural Computation (2002)
    - What Learning Systems do Intelligent Agents Need? Complementary Learning Systems Theory Updated, Dharshan Kumaran, Demis Hassabis, James L. McClelland, in Trends in Cognitive Sciences (2016)
- Related datasets and environments:
    - OpenAI Gym
    - DeepMind Lab
    - Contextual RNN-GAN
    - Bringing Semantics Into Focus Using Visual Abstraction
      Learning the Visual Interpretation of Sentences
    - AI2 datasets
- GoodAI Framework: http://www.goodai.com/framework

Appendix A: Micro-tasks overview

A.1 Introduction

A.1.1 Purpose of micro-tasks

The aim of the first Challenge round is to build an agent that exhibits gradual learning. This will be tested by providing to the agent a set of tasks of gradually increasing complexity in the provided environment. The tasks come in two groups, as simpler “micro-tasks” and as more advanced “mini-tasks”.

The mini-tasks, based directly on the CommAI-mini set, focus on simple grammar processing and deciding language membership problems for strings. The mini-tasks are simple for an educated and biased human, but they are tremendously complex for an unbiased agent that receives only sparse reward.

Therefore we decided to insert simpler tasks, which we refer to as “micro-tasks”. The purpose of micro-tasks is to allow the agent to pick up biases necessary for discovering the solutions of mini-tasks in reasonable time, under the assumption that the agent is capable of gradual learning. The “picking up of biases” is just another name for learning useful skills.

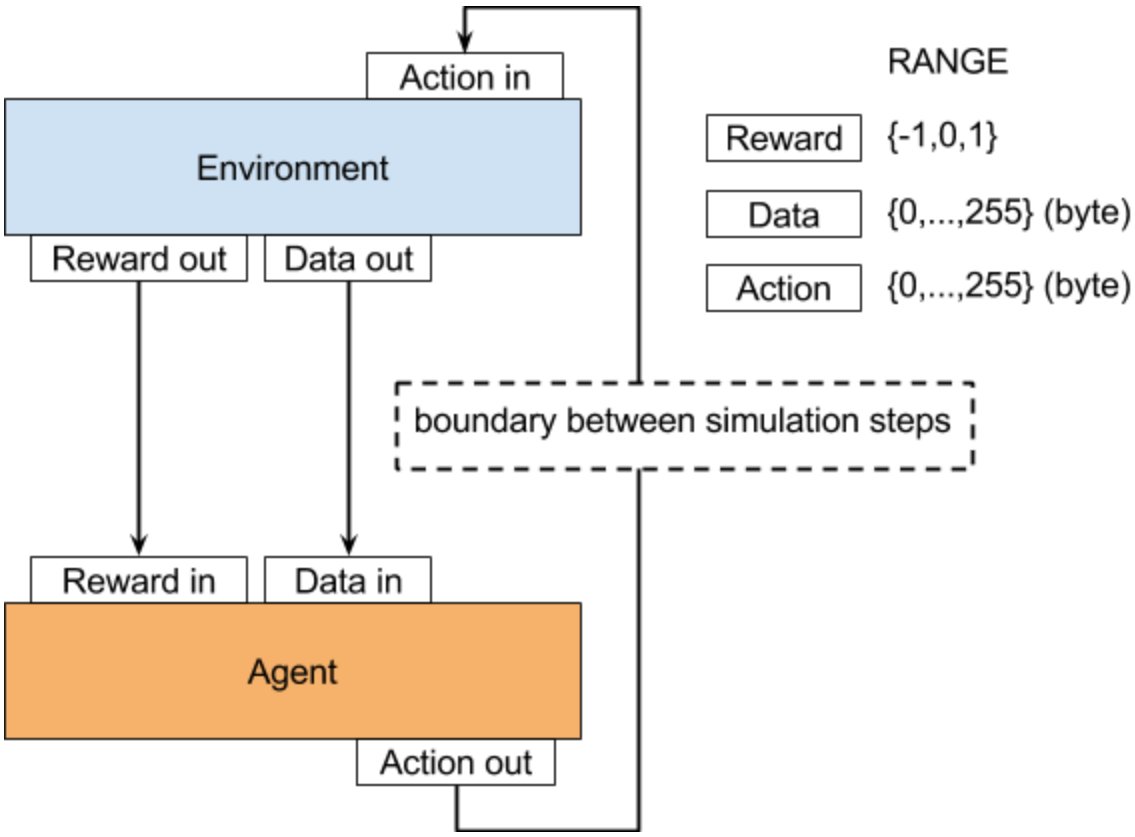
Decomposing mini-tasks into simpler problems yields a number of skills that the agent should be taught. For each such skill, there is a separate micro-task. The micro-tasks build on each other in a gradual and compositional way, relying on the assumption that the agent can make use of them efficiently.

We are aware that the curriculum of micro-tasks that we present in this document is by far not the only one possible for testing and guiding the development of gradually learning agents. We wanted it to serve as an inspiration, but at the same time make it usable for actual training of a gradually learning system. If anyone trying to solve mini-tasks finds out the curriculum presented here is not suitable for the system they are building, they can come up with a curriculum of their own or extend the provided curriculum.

One could argue that the micro-tasks (and even the mini-tasks) are too simple and can be solved by hard-wiring the necessary knowledge into the agent. This is indeed true. However, such a solution is not desirable and goes against the spirit of the challenge - to learn new skills from data. The challenge will prevent any hard-wired solution by using hidden evaluation tasks which are different from the public training tasks. A hard-wired solution that does not exhibit gradual learning should fail on these tasks.

A note should be taken that natural language processing (NLP) is not necessary for solving the micro-tasks and the mini-tasks. Nor will it be necessary for solving the evaluation tasks.

A.1.2 Environment-agent interface



The interface between the agent and the environment is depicted in the above Figure.

The environment sends reward (-1,0 or 1) and data (one byte) to the agent and receives the agent's action (one byte) in response. This happens in a continuous cycle. During a single **simulation step**, the environment processes the received action from the agent and sends reward with new data to the agent; the agent processes this input and sends an action back to the environment.

Very specifically, the environment sends 8 bits of data to the agent at every simulation step and receives 8 bits of data from the agent at every simulation step. This is different from the original CommAI-env, which sends only a single bit instead of a full byte at a single simulation step. Although this makes the interface a little less flexible than in the original version of CommAI-env, it should at the same time make the agent's communication more interpretable and it should reduce the complexity of the presented problems.

Mind that the environment is not sending any special information about what task the agent is in at the moment. The environment seems as a continuous stream of bytes (and rewards) to the agent.

A.1.3 Format of the micro-task specifications

The micro-tasks described in this document share a common description format. First, there is a textual description of what the task is teaching. Next, there is an example of input, output and reward flowing between the agent and the environment during training. Last, there is optionally a description of a success criterion that indicates whether a single task instance has been solved successfully.

In formal contexts or to remove ambiguity, terms "**micro-task set**" and "**micro-task instance**" are used instead of informal "micro-task", which can represent both. A single micro-task set consists of many different micro-task instances. The examples which demonstrate the micro-task sets always show a single micro-task instance.

Micro-task instance example format (from micro-task set A.2.2):

```
Input : waaaaaaaaait for it.     good job!
Output: fedcbbbbbbbbbxbbbybb.z   bbbb bbb!
Reward: ----++++++++++-+++-+++++++++++++
```

Figure 2: Example of a micro-task instance from micro-task set A.2.2.

The examples should be read in columns; the top row contains the input byte from the environment, the row below contains the reaction byte from the agent and the bottom row contains the reward that the environment sends to the agent for its action. The time in the examples flows from left to right. The following figure depicts these principles.

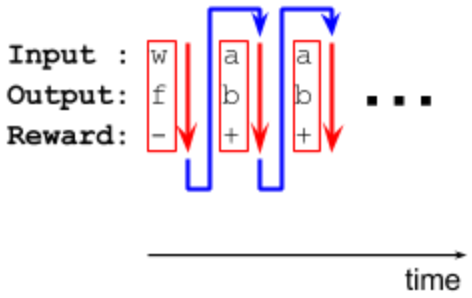


Figure 3: How to read examples of micro-task instances.

The micro-task examples use a plus sign to depict a +1 reward and a minus sign to depict a -1 reward. When the reward is zero, it is indicated with a space.

As was said above, the examples always show a single micro-task instance. A micro-task set consists of multiple (auto-generated) instances. To explain better, the following Figure depicts another instance from micro-task set A.2.2:

```
Input : try again and see if you succeed. nice!
Output: bbb abcde fgg ggg gg ggg ggggggg. gggg!
Reward: ---+-----+++++++++++++++++++++++++++++
```

Figure 4: Example of another instance from micro-task set A.2.2.

Micro-task set 2.2 requires the agent to find which symbol the letters of alphabet are mapped to and output that symbol upon seeing a letter of alphabet. The mapping changes between task instances (notice that in the first example, the letters were mapped to symbol ‘b’, whereas in the second they were mapped to symbol ‘g’).

A.1.4 Success criteria

There are different ways how it can be decided that the agent has learned to solve some task. A simple criterion that we chose as the default one examines the number of successful attempts *at task instances* in a row. If the agent exceeds a threshold number of successfully solved task instances in a row, we declare that it learned to solve the task and it can move on to another task.

The success criteria for task instances are different for each task, but have one thing in common: the agent is required to use the available information as efficiently as possible to solve the task instance as quickly (=with as little data) as possible. For instance, in the task illustrated in the examples above (Figure 3 and 4), a sufficiently capable agent will eventually learn that any non-alphabet-letter character is always mapped to itself and every letter of alphabet is mapped to the same, albeit randomly chosen symbol. It should therefore need at most 26 steps (excluding the steps where the world presents non-alphabet-letters) to find the correct alphabet symbol given a new task instance (‘b’ or ‘g’ in the examples above) and then apply this symbol correctly for say 10-20 more steps.

We are aware that an agent that is still learning to solve a task cannot solve a task instance within a strict time-step threshold as described above. This is why the agent is allowed to try to solve the task instance for some additional time (and it will receive rewards/punishments for its attempts on it), although any solution found in this additional time will not be counted as a next task instance solved in a row.

More details on how instance success is calculated are available at the descriptions of tasks A.2.1 and A.2.2, especially in the subsubsection instance success criterion. The process of evaluating instance and task success is also described in detail in the readme file at the github repository of the challenge.

A.1.5 Training / testing split

We are not providing any clear boundary between training and validation (or testing) data for gradual learning. It is up to the authors of a gradual learner AI to decide which of the following tasks they want to use for training and which (if any) they will use for validation (testing).

A.1.6 Evaluation

In a standard machine learning problem, the testing data are drawn from the same distribution that the training data came from. In case of the gradual challenge this principle can be used, but it has to be adapted.
An agent trained on one curriculum cannot be evaluated on the tasks from the same curriculum to test gradual learning. A different curriculum is necessary.
The curricula appropriate for a gradually learning agent themselves form a distribution from which a training curriculum and an evaluation curriculum should be drawn. The mini-tasks and micro-tasks used in the challenge form together one sample from the distribution. But given a single sample, the properties of the distribution are hard to guess. We therefore provide examples of most difficult tasks from alternative curricula from the same distribution. You can expect the evaluation curriculum to lead to tasks of similar complexity.

A.1.6.1 Alternative curriculum #1

A.1.6.1.1 Hardest task:

Compute the value of an arithmetic expression that uses parentheses, plus, minus, times and integer division operations and contains initialized variables. The numbers occurring in the expression are encoded in hexadecimal format; the result should be printed in decimal format and preceded by number of digits required to write down the result. The feedback can require some processing on agent side.

```
Input:  Evaluate: foo=4 ce; (foo+1a5eb)*a+1=        good if a==5+5
Output:                                    7 1092411
Reward:                                           +
```

A.1.6.2 Alternative curriculum #2

A.1.6.2.1 Hardest task:

In an environment with rules explained and taught in the previous tasks of the curriculum, follow a conversation about the environment. The environment can for instance resemble the notorious Blocks World. However, it is not the intention of the authors of the challenge to teach or require communication in natural language. A simplified language with a small grammar would suffice (see the example below).

```
Input:  see: table desc ribe: object?                CUBE yes LARGE yes GREEN
yes
Output:                             CUBE LARGE GREEN !
Reward:                                             +
```

A.2 Specification of the tasks

The following list describes micro-tasks that teach the agent skills necessary for the mini-tasks.

A.2.1 Outputting a subset of ASCII characters only (a-zA-Z0-9 ,.!;?-)

Make sure that the agent is outputting only allowed ASCII characters. If we fix the 8-bit I/O, this could be hardcoded in the agent. If it's to be learned, a task instance will choose a character at random that the agent should always output. The chosen character should belong to the set of allowed ASCII characters.

```
Input : find the allowed character. once you find i t, repeat it
Output: qwertyuiopaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaa
Reward: ----------++++++++++++++++++++++++++++++++ ++++++++++++
```

A.2.2 Learning a mapping between symbols from input to Output:N to 1

The agent has to discover a mapping that exists between each symbol on the input and a required output symbol. To make the task easier, there are several groups of symbols on the input and all symbols within one group are mapped to the same character.

```
Input : waaaaaaaaaait f or it.     good job!
Output: fedcbbbbbbbbbxbbbybb.z    bbbb bbb!
Reward: ----++++++++++-+++-+++++++++++++++
```

A.2.3 Learning a mapping between symbols from input to Output:1 to 1

The agent has to discover a mapping that exists between each symbol on the input and a required output symbol. The mapping is different between task instances.

```
Input : waaaaaaaaaait for it.     good job!
Output: xbbbbbbbbbbop tre op.     krrg qrm!
Reward: ++++++++++++++++++++++++++++++++
```

NOTE: to find a mapping for the whole alphabet using reward signal only for feedback, this task can take 525=(26-1)x(26-1) steps to solve even for an ideal agent.

A.2.4 Learning to copy input to output

Learning to repeat input on the output. This is a special version of the previous task.

```
Input : waaaaaaaaaait for it.     good job!
Output: brmblsaaaaait for it.     good job!
Reward: -----++++++++++++++++++++++++++++++
```

NOTE: this task is solvable by a lookup table - any evaluation should treat the agent's performance on it accordingly.

A.2.5 Learning correct output format from the input (feedback occurs on input)

A.2.5.1 One byte for input, output and feedback

After an input byte is presented, the agent is expected to output the correct byte in response. Regardless of the agent's output, the environment will respond with the byte that was expected. The expected byte depends on the previous input byte. After the feedback is presented to the agent, the environment presents a new input byte and the cycle repeats..

If the agent does not understand yet that it should be outputting spaces while not responding, it can be given negative reward for speaking (=outputting a non-space) when not asked to speak.

In the following task instance example, the agent should learn to output a '0' when it sees a '0' on input, then read the feedback and output a space (because any other character is rewarded negatively). The agent should also learn to output a '1' when it sees a '1' on input (and again output a space after reading the feedback).

```
Input : 0011001100001
Output: 53248 1 1 0
Reward: ----- + - + -
```

Figure 5: Task instance #1 of task A.2.5.1.

To make the principle of this task clear, here is another task instance. In this case, the agent should learn to respond to character '0' with a '6' and to character '1' with a '2'. When the agent learns to respond correctly, it effectively learns to predict the future.

```
Input : 061206120606
Output: 0 2 8 2 1 6
Reward: - + - + - +
```

Figure 6: Task instance #2 of task A.2.5.1.

Any two task instances share the one byte property, but they can otherwise map different inputs to different required outputs. The following tasks extend and build upon this first task and provides further explanation on instance switching.

A.2.5.2 Feedback separator

Same as "one byte for input, output and feedback", but the **input and feedback** are **separated** by a special symbol ' ; ' .

```
Input : 00;11;00;11;00;00;...
Output:    2  8  1  1  0  ...
Reward: -  -  -  +  -  +  ...
```

Recall that the task is presented in multiple instances to the agent. The agent does not receive any indicator that would tell it that task instances have switched - it has to find out this on its own.
The following Figure shows what a task instance switch will look like to an agent. Note that the agent that solves the task instance correctly must adapt quicker to the task switch than the agent from the Figure, who made unnecessary mistakes upon seeing symbol '0' (replying with '8' and '0' instead of '5'):

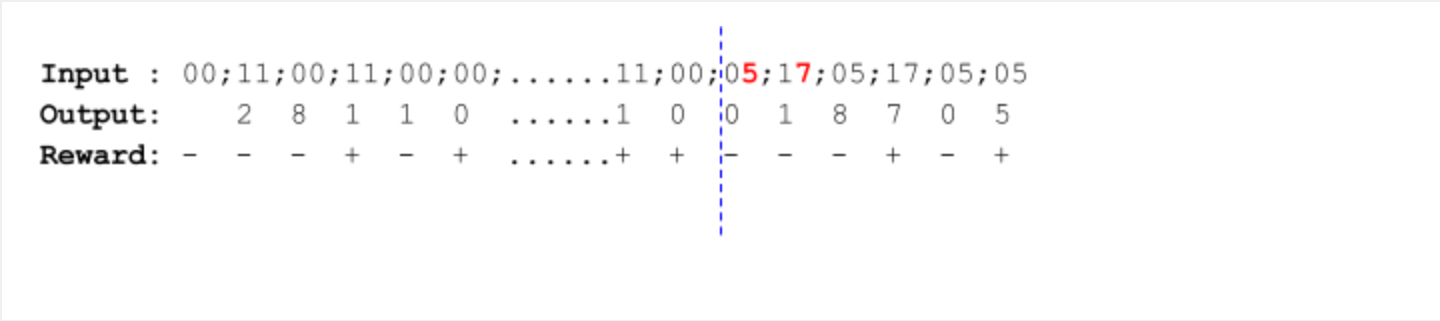


Figure: Illustration of a switch from one task instance to another. The moment of the switch is highlighted with a blue dashed line. In the previous task instance, correct output for input 0 is 0 and 1 for input 1. In the next task instance, the correct output switches to 5 for 0 and 7 for 1.

A.2.5.2.1 Instance success criterion

The agent successfully solves a task instance if, for each input (not feedback) digit that is shown for the second time, third time, n-th time, it predicts the correct feedback digit. If the agent makes a mistake, the environment keeps presenting data from the current task instance until a timeout occurs or the agent finally solves the instance (although such solution is not considered a success).

A.2.5.3 Input separator

Same as "feedback separator", but the **input and output** are **separated** by a special symbol ' . ' .

```
Input : 0.0;1.1;0.0;1.1;0.0;0.0
Output:      2   8   1   1   0
Reward:  -   -   -   +   -   +
```

A.2.5.4 Multi-byte output and feedback

Same as "input separator", but the expected **output and feedback** consist of **two** characters.

```
Input : 0. 00;1. 11;0. 00;1. 11;0. 00;0. 00;
Output:  1     1     0     11    10    00
Reward:   -     -     -     +     -     +
```

A.2.5.5 Output terminated with a period

Same as “multi-byte output and feedback”, but the expected **output and feedback** consists of **two** characters, of which the second is always a period.

```
Input : 0. 0.;1. 1.;0. 0.;1. 1.;0. 0.;0. 0.;
Output:  1     1     0     1.    1.    0.
Reward:   -     -     -     +     -     +
```

A.2.5.6 Output terminated with a period without period in feedback

Same as “output terminated with a period”, but the period is not being shown in the feedback **when the response is correct**. Once the agent is reliably ending its outputs with a period, the feedback can drop the period even when the answer is incorrect. When the agent’s answer starts missing a period, however, the feedback should again include it.
Other tasks might be based on this task rather than “output terminated with a period” task.

```
Input : 0. 0.;1. 1.;0. 0.;1. 1;0. 0.;0. 0;
Output:  1     1     0     1.    1.    0.
Reward:   -     -     -     +     -     +
```

A.2.5.7 Multi-byte output and feedback of variable length

Same as "output terminated with a period", but the expected **output and feedback** can consist of **one or two** characters (excluding the terminating period). The feedback begins immediately after the agent outputs a period or after its output exceeds a limit length.

```
Input : 0. 0.;1. 11.;0. 0.;1.  11.;0. 0.;
Output:  1.    1.     1.    11.     0.
Reward:   -     -      -      +      +
```

A.2.5.8 Multi-byte input

Same as "output terminated with a period", but the **input** consists of **two** characters plus the '.' symbol.

```
Input : 00. 0.;10. 1.;01. 0.;10. 1.;00. 0.;
Output:   1.     0.     1.     1.     0.
Reward:    -      -      -      +      +
```

A.2.5.9 Multi-byte input of variable length

Same as "output terminated with a period", but the **input** consists of **one or two** characters plus the **'.'** symbol.

```
Input : 00. 0.;1. 1.;01. 1.;10. 1.;00. 0.;
Output:   1.    0.     0.     1.     0.
Reward:    -     -      -      +      +
```

A.2.5.10 Multi-byte feedback with unimportant sections

Same as "output terminated with a period", but the **feedback** consists of **two** characters plus a **'.'** symbol, out of which **the first one can be ignored**. The ignored character can be random for each question.

```
Input : 0. 00.;1. 01.;0. 10.;0. 10.;1. 01.;
Output:  1.     0.     1.     0.     1.
Reward:   -      -      -      +      +
```

The aim for this task is to start teaching the agent that there may be multiple different ways how to interpret the feedback; understanding it as an exact copy of the correct answer might not be always the right thing to do. Consider the following dummy CommAI-env task:

```
Wrong! Correct output was: True.; Tell me how many pears you have.
```

The feedback in the above task instance is hidden after a sentence that can be largely ignored (highlighted in red). The agent should not think that the feedback is always everything that occurs before a ';' symbol.

A.2.5.11 Multi-byte feedback with unimportant sections of variable length

Same as "output terminated with a period", but the **feedback** consists of **one or two** characters plus a **'.'** symbol; if it's two characters, then the first one can be ignored.

```
Input : 0. 10.;1. 01.;0. 0.;0. 00.;0. 0.;1. 11.;
Output:  1.     0.     1.    0.     0.    1.
```

```
Reward:    -       -       -      +       +      +
```

A.2.5.12 Multi-byte output and feedback of higher length

Same as "output terminated with a period", but the expected **output and feedback** consists of either **three or four or five** characters plus the '.' symbol. Showing an example for the case of three characters.

```
Input : 0. 101.;1. 010.;0.   101.;1.   010.;0.   10 1.;
Output:  1.       1.       111.       010.       101.
Reward:   -        -          -          +          +
```

A.2.5.13 Multi-byte input of higher length

Same as "output terminated with a period", but the expected **input** consists of either **three or four or five** characters plus the '.' symbol. Showing an example for the case of three characters.

```
Input : 000. 0.;101. 1.;010. 0.;101. 1.;000. 0.;
Output:    1.       0.       1.       1.       0.
Reward:     -        -        -        +        +
```

A.2.5.14 Multi-byte feedback of higher length with unimportant sections

Same as "output terminated with a period", but the **feedback** consists of either **three or four or five** characters plus the '.' symbol, out of which all but the last one can be ignored. Showing an example for the case of three characters.

```
Input : 0. 010.;1. 011.;0. 100.;0. 110.;1. 001.;
Output:  1.       0.       1.       0.       1.
Reward:   -        -        -        +        +
```

A.2.5.15 Multi-byte output and feedback of variable length

Same as "output terminated with a period", but the expected **output and feedback** consist of **one to five** characters.

```
Input : 0. 101.;1. 10.;0.   101.;1.  10.;0.   101.;
Output:  1.       1.      111.       10.      101.
Reward:   -        -         -         +         +
```

A.2.5.16 Multi-byte input of variable length

Same as "output terminated with a period", but the expected **input** consists of **one to five** characters.

```
Input : 001. 1.;11. 0.;0. 1.;001. 1.;0. 1.;
Output:    0.     1.    0.      1.    1.
Reward:     -      -     -       +     +
```

A.2.5.17 Multi-byte input, output and feedback of variable length

Same as "output terminated with a period", but the expected **input, output and feedback** consist of **one to five** characters.

```
Input : 001. 11.;11. 0.;0. 1.;001.  11.;0. 1.;
Output:    0.      1.    0.      11.     1.
Reward:     -       -     -       +      +
```

A.2.5.18 Multi-byte input, output and feedback of high (and variable) length

Same as "multi-byte input, output and feedback of variable length", but the expected **input, output and feedback** consist of **one to ten** characters.

```
Input : 001101. 11.;11. 0.;0. 1.;001101.  11.;0. 1. ;
Output:       0.      1.    0.          11.     1.
Reward:       -       -     -           +      +
```

A.2.6 Learning useful sequences (a.k.a. English words)

A prerequisite task is “multi-byte input, output and feedback of high (and variable) length”. In this task set, the learner will learn to answer same kind of problems as in the prerequisite task, but the sequences will be limited to English words. The intent is to start familiarizing the learner with English words so that the learner can quickly identify a known sequence without dedicating too much processing power/memory to recognizing it and storing it.

```
Input : random_map: apple.    house.;random_map: ap ple.     house.;
Output:                  0011.                          house.
Reward:                     -                              +
```

NOTE: this task is solvable by a lookup table - any evaluation should treat the agent's performance on it accordingly.

A.2.7 Repeat a provided word to output.

This group of tasks aims to teach the agent to copy a provided word to output. As a prerequisite, the agent is expected to be able to solve tasks from task group A.2.5. The tasks start from single symbols and gradually build up to full words or even sequences of words.

A.2.7.1 Repeat a single-letter word to output.

Notice the consistent use of command "say". This command will be used for asking the agent to output some text from this task onwards.

```
Input : say a. false. a; say a. correct. a.; say b.  ...
Output:      0.               a.
Reward:       -                +
```

A.2.7.2 Repeat a provided word to output.

Moving from a single letter to a single word.

```
Input : say hello. false. hello.; say hi. false. hi .; say hi.
Output:          o.                    I.                   hi.
Reward:           -                     -                    +
```

NOTE: this task is solvable by a lookup table - any evaluation should treat the agent's performance on it accordingly.

A.2.7.3 Repeat a provided sequence of words to output.

Moving from a single word to a sequence of words.

```
Input : say: hi world.  false. hi world.; say: hi w orld.
Output:              hi.                                hi world.
Reward:               -                                      +
```

A.2.8 Understanding redundancy of spaces

In some tasks, there might be multiple meaningless spaces in a row in input with no impact on result. This task should help the agent understand this fact - the agent should learn to ignore redundant spaces.

As a prerequisite, the agent is expected to be able to solve tasks from task group A.2.5.

```
Input : say:    a  b   .    correct! a b .; say: …… .
Output:                a b .
Reward:                   +
```

During implementation, this task might need to be split into multiple sub-tasks to teach the agent about redundancy of a single space, two spaces, three spaces etc., eventually extending to any number of spaces.

A.2.9 Basic manipulation with sequences

To verify the agent is able to correctly separate bytes, we can let it add a space between all n-tuples of characters. This is not necessary if byte parsing is hardcoded in the agent. But this is very simple task which can be introduced right after a repetition task to test memory control flexibility. To make learning easier, it provides **negative reward immediately**. It may provide positive reward immediately too.

Note: the actual code will start with shortest sequences and only gradually move to longer ones.

A.2.9.1 Spell a single-letter word

This task is the same as the following (spell a word), but the words are one character long.

A.2.9.2 Spell a word

Take the provided word and print it to the output one character at a time. Follow each character with a space character.

Ver. 1 (negative reward immediately)

```
Input : spell: mono.        good job; spell: stereo .    wrong! s...
Output:            m o n o .                          s te
Reward:                    +                             -
```

Ver. 2 (both negative and positive reward immediately)

```
Input : spell: man.  wrong! m a n .;spell: moss.         good job!
Output:           m                            m o  s s .
Reward:           ++-                          ++++ +++++
```

A.2.9.3 Interleave a word with a symbol

This task is similar to “spell a word”, but the agent separates the word’s characters with a provided symbol. Moreover, the last character is not followed by the provided symbol.

```
Input : interleave: man by -.     Good! m-a-n.
Output:                     m-a-n.
Reward:                          +
```

A.2.9.4 Other suggestions:

1. Output the shorter (or longer) word of the two provided words

2. Output the shortest (or longest) word of the provided words
3. Output a sequence of words except a given word
4. Remove occurrence of a word from another word, where it occurs as a substring
5. Find occurrence of a word in a another word, where it occurs as a substring (true/false or speak while the environment is outputting another word)
6. Remove all occurrences of a word from another word, where it occurs as a substring
7. Find all occurrences of a word in a another word, where it occurs as a substring

A.2.10 Perform multiple commands

The agent must be able to comprehend multiple commands at once

```
Input : say: a say: b. wrong! a b.;say: a say: b.    good job! a b.
Output:              b.                          a b .
Reward:               -                              +
```

A.2.11 Memory for at least 2 pieces of data

The agent needs to perform some kind of mental operation which requires it to hold two pieces of data separately. Mental operations are: reverse, concatenate, interleave. The mental operation to perform is presented to the agent only after the two pieces of data so that it really has to store the pieces in its memory.

```
Input : say: ab say: cd reverse:.     good job!
Output:                         cd ab.
Reward:                              +
```

```
Input : say: ab say: cd concatenate:.    good job!
Output:                               abcd.
Reward:                                   +
```

```
Input : say: ab say: cd interleave:.    good job!
Output:                              acbd.
Reward:                                  +
```

Note: an extended version of this task could also chain the mental operations, for instance reverse and concatenate.

A.2.12 Identify relationship between parts of input

This is a direct continuation of the previous task (“memory for at least 2 pieces of data”). In the previous task, the agent did not need to change one piece of data using the other piece of data. In this task, the agent needs to understand that sometimes correct response needs combining information from different sources. The agent will be asked to combine the two provided pieces of information (“ordered” set union, “ordered” set difference).

```
Input : say: ab say: b union:.  good job!
Output:                       ab.
Reward:                         +
```

```
Input : say: abc say: b exclude:.  good job!
Output:                          ac.
Reward:                            +
```

A.2.13 Learning logical operations useful for mini-tasks

A.2.13.1 Learning basic logical operations for mini-tasks (and, or, not)

An agent told to say “X and Y” should reply “XY.” (same as “say X say Y concatenate”).

```
Input : say: a and b.  good job!
Output:             ab.
Reward:               +
```

An agent told to say “X or Y” should reply “X.” or “Y.” or “XY.” or “YX.”

```
Input : say: a or b. good job!
Output:            a.
Reward:             +
```

An agent told to say “anything and not X” should say something different from “X.”

```
Input : say: anything and not a. good job!
Output:                         b.
Reward:                          +
```

An agent told to say “X or Y but not Z” should reply the same as when asked to say “X or Y” when Z is different from X and Y. Otherwise it should reply “X.” if Y is equal to Z (and vice versa).

```
Input : say: a or b but not a. good job!
Output:                       b.
Reward:                        +
```

A.2.13.2 Learning basic logical operations for mini-tasks 2 (multiple operations)

This test is made of logical expressions containing multiple operations but no negation.

```
Input : say: a and b and c and d and b.     correct !
Output:                                 abcdb.
Reward:                                      +
```

```
Input : say: ab or ef or c or de.   correct!
Output:                          abc.
Reward:                             +
```

A.2.14 Alphabet ordering

The agent is taught the order of symbols in alphabet. Although not a useful task of itself, it is useful for advanced tasks.

```
Input : after d comes what:. good job! e.; after g  comes what:.
Output:                      e.                                  h.
Reward:                       +                                   +
```

NOTE: this task is solvable by a lookup table - any evaluation should treat the agent's performance on it accordingly.

A.2.15 Learning useful sequences of sequences

A prerequisite task is "learning useful sequences (a.k.a. English words)". This task teaches the agent a fixed mapping from one word to another, which can then be used for forming sequences of words. It is similar to the task that teaches alphabet ordering. The agent will succeed in this relatively simple task when it finds the correct mapping between numbers.

```
Input : say next after: one.   two.;after one comes  what:.   two.;
Output:                    one.                            two.
Reward:                       -                              +
```

NOTE: this task is solvable by a lookup table - any evaluation should treat the agent's performance on it accordingly.

A.2.16 Unchanging parts of input as a context for the algorithm

After agent learned a few tasks, it should be able to recognize which task it is in, depending on the question (a.k.a. the context) and respond correctly. This task set can be inserter repeatedly with different questions (contexts).

```
Input : say: a. good job! a.; after d comes what:.  good job!
Output:       a.                                    e.
Reward:        +                                     +
```

A.2.17 Hierarchical memory / hierarchical processing

There is no separate task for testing hierarchical reasoning. The tests of hierarchical reasoning are instead spread throughout the tasks. For instance, tasks A.2.11 (“identify relationship between parts of input”) and task A.2.8 (“basic manipulation with sequences”) show hierarchical properties.

A.2.18 Combining skills / compositionality

All the above tasks actually test combining skills, so there is no separate task for this skill at the moment.

A.2.19 Understanding synonyms

The agent should learn modifications of already known concepts faster. So when you give it a similar task with few words changed, it should be able to learn the new task faster.

```
Input : write: a. correct; print: b. correct; print : c. correct;
Output:         a.                 b.                  c.
Reward:          +                  +                   +
```

```
Input : say: a and b.  good job; say: c & d.  corre ct;
Output:             ab.                    cd.
Reward:               +                      +
```

NOTE: this task is solvable by a lookup table - any evaluation should treat the agent’s performance on it accordingly.

A.2.20 Hint understanding

To understand synonyms better, we can introduce relation “is as” to help the agent connect a correct skill with a new one. In this configuration, the first part of a sentence describes how to change a skill while the second part is testing the changed skill. So this together with synonyms understanding task forms a test of

meta-learning - if the agent performs better in this test than in the synonyms understanding (which shows no hints), it could be seen as a sign of a meta-learning ability.

```
Input : print is as say - print: panda.     correct ;
Output:                                  panda.
Reward:                                        +
```

```
Input : & is as and - say: a & b.  correct;
Output:                           ab.
Reward:                             +
```

A.3 Illustration of micro- and mini- tasks

The following figure illustrates the challenge tasks and the dependencies between them.

Full resolution diagram available at:
https://drive.google.com/file/d/0B820uHFOHp0mbnhtbWdRNVJRNjg/view?usp=sharing