Submission form

Submission category: **Both / Quantitative and qualitative prize**
1) Author: **Andrés del Campo Novales**
2) Evaluation hardware: **CPU-bound**
3) Target operating system: **Windows**

Agent executable

Running the agent

No extra set up steps are necessary other than **running Learner.cmd**.

Compiling the agent

The agent has been developed in Visual Studio 2017, as a solution composed by a Console Application (CSLearner) and a Class Library (CSLearnerLib), both with .NET 4.5.2. A third project contains unit tests.

NetMQ was added as external reference and is required to compile the project. Refer to: https://www.nuget.org/packages/NetMQ/ on how to install it in case the zipped project sources are not sufficient.

Design

This agent implementation follows a heuristic approach which aims to provide the agent with enough building blocks (abstraction rules) to abstract knowledge based on seen examples. Note that the agent is not bound to any specific input language or symbol even though it may appear hardcoded at first sight.

It is composed of:

- **Rule engines:** Allow the agent to abstract, store, detect future occurrences and apply generic rules. These are *the core learning components of the agent*, detailed in the next section.
- **Syntax and State:** These components handle the separation of inputs from feedbacks and detecting when to output, correct if needed, etc. While they are bound to a specific set of syntax rules, they allow flexibility on the character(s) used to separate requests or to specify when to answer, the length of the feedback or any specific word or sentence that is merely providing feedback to the agent and not being part of the answer (i.e. “wrong!”), etc.
- **Learner:** Agent library entry point that handles the input and rewards to provide the output. It coordinates the prioritization between the rule engines to provide the best answer, triggers the abstraction and learnings from feedback, decides when to course-correct, and communicates with the syntax and state components.

Rule engines

These rule engines are the core of the agent, and allow the agent to gradually learn skills from examples.

- **Mapping rules:** A short term memory that maps directly inputs to outputs, which can be forgotten in bulk when proven incorrect.
- **Word generic rules:** Functionality that allows the agent to abstract, store and use rules based on patterns in which words repeat between different inputs and outputs. An example of this may be:

  ```
  Reverse: aaa bbb ccc        ->      ccc bbb aaa
  Reverse: black white green ->       green white black
  ```
  Generates a rule like:
  ```
  Reverse #1 #2 #3            ->      #3 #2 #1
  ```

  The abstraction engine traverses the last input and output and matches it with the previous knowledge to verify what is constant and what varies, to abstract a pattern if it exists.

This rule engine can also learn from word composition too:

And: Word1 Word2 Word3 -> Word1Word2Word3

- **Character generic rules:** Similar to the Word generic rules, these heuristics allow the agent to abstract patterns between input and output based on individual characters. These have proven *extremely resourceful in learning anything from concatenation, reversal, interleaving, even and, union, or…* just by using this simple abstraction mechanism, especially in combination with the size abstraction described in the next point. Examples:

  Interleave: test @ -> t@e@s@t
  Interleave: mark # -> m#a#r#k
  Interleave: #1#2#3#4 #5 -> #1#5#2#5#3#5#4

- **Character generic rules with size abstraction:** The character generic rules have a weakness. They are bound by the size of the word in the input. For example: Learning Interleave with 4 characters will not suffice to answer an interleave with 5. This abstraction engine analyzes the character rules known looking for variations of one character that show the same transformation patterns. For example:

  Interleave: #1#2#3 #4 -> #1#4#2#4#3#4
  Interleave: #1#2#3#4 #5 -> #1#5#2#5#3#5#4

  And abstract a growth rule like

  Interleave #1 #3 -> #1 with growth of #3#2 *to the right* -in which #2 would be the next character.
  This engine can then create and apply rules of any size that match the pattern, when needed.

- **Compound rules:** Some of the previous rule engines have been enhanced to allow matching and applying them multiple times in the input, usually in basic concatenation (Rule1 Rule2 -> Output1 Output2, in which the first rule might be Reverse and the second Interleave, for example). Those are attempted to be applied whenever the individual patterns are not found.

- **Math rules:** After considering more generic approaches, and due to the lack of time, I decided to go for a recognition of pre-known operations (addition, subtraction, multiplication, division -easily expandable), which allows the computer to use what it already knows without relearning the internals of the operation. It is flexible enough to learn anything like

  A + B -or any other symbol
  ADD A B -or any other words / formatting
  SUM(A,B)

  and can work in several numeric bases both in input and outputs.
  The math engine can also apply operations in a rolling basis (a + b * c / d…) though it is limited at the moment and it may not be able to detect more advanced inputs.

Performance and conclusions

Unfortunately, I discovered this challenge just with 2 months left to the deadline, and while the agent passed the training tasks after 4-5 weeks of development, it is unlikely to pass the evaluation tasks as it would still require further abstraction building blocks to accomplish the deeper learning required for some of them. The agent can still be expanded in many ways otherwise. I thought that after this short but very intense challenge, it would be useful to deliver this agent even if incomplete.

I also found the character rules and size generalization as apparently simple concepts though still very, very powerful in the outcomes they could accomplish to the point of surprising me with the training tasks it could solve without having been explicitly programmed to do so.