Practical assignments - Language evolution, language acquisition and language grounding

Paul Vogt
p.a.vogt@uvt.nl

5–9 May 2008, Helsinki

1 Introduction

There are two possible assignments in this course. One is to program your own system, which will be discussed in the next section. The other is based on using the THSim toolkit, which will be discussed in section 3. If you have programming skills, I advice you to to do the programming assignment, because I believe that will be more instructive and more fun. In both case, the aim is to conduct a small research and write a small report (4 to 6 pages) on this research. You will be credited based on this report.

I expect you to work in small groups of two persons, because that allows you to discuss issues with each other, which I think is very instructive. The assignments provided in this document are not to be taken too strict, but simply as guidelines. I expect you to take your own initiatives into this assignment as much as possible.

The THSim toolkit, data set and script mentioned are available from the course home page: http://ilk.uvt.nl/~pvogt/helsinki.html

Most importantly, enjoy what you are doing and enjoy Helsinki.

2 Programming assignment

The idea of the programming assignment is to write a small multi-agent system that can simulate some aspects of language evolution, language acquisition and language grounding. The goal is to carry out a small research study near the end of the week and write a short report.

Below follows a number of steps that you should follow in order to get at a stage where your are ready to perform your small research. Before you start, please read the entire assignment, so that at each step you can already take into account that you have to extend your program. Use the programming language you are most familiar with and that is available in the lab.

2.1 Implement a simple naming game

The implementation should contain the following elements:

Simulation-engine The simulation engine is the heart of the system. It initialises a set or list of agents (described below), which form the population. Take care that you can easily change the population during the simulation. After the initialisation, the engine goes into a loop to play a given number of naming games (see below).

Agent You need just one function (ideally a class) for all agents. The agent should have the following data structure and procedures:

Lexicon The lexicon is a data structure, which is a two dimensional matrix of real values (these are the scores σ_{ij} indicating the strength of the association between word w_i and meaning m_j). The two dimensions are *words* and *meanings*. For now we shall assume that meanings are integers (later we will have them grounded). The words are random strings of characters. Note that you either reserve enough space for your lexicon (the number of words are typically much more than the number of meanings), or make it dynamically extendable.

Production module The production module searches for a given module the word that best expresses this meaning. If a meaning has no word yet, a new word is created as a random string and added to the lexicon. (This word creation occurs with a probability, which is usually set to 1, but can be used as a parameter of investigation.)

Interpretation module The interpretation module searches for a given word and context (see below) the meaning that best fits the word and is in the context.

Adaptation module Here the scores σ_{ij} are adapted and – if necessary – new words are acquired by adding them to your lexicon. Other learning mechanisms could be added at a later stage.

Play game Select 2 agents randomly from population, assign one of them the role of speaker, the other the role of hearer. Select a random meaning as the *target* of the game (use integers $1, \ldots, M$, where M is the number of meanings). Select $C - 1 << M$ additional meanings that can form the *context* of the game. Let the speaker produce an utterance to convey the target and the hearer interpret this utterance, given the context. Have both agents adapt their lexicons. Determine whether the game is successful and output data concerning the game (e.g., game number, agents, meaning, utterance and success). TIP: write 0 or 1 indicating failure or success. When you put this data in a file with two columns, where the first is the game number and the second the success, then you can then use the program `calc_average_success` in the scripts directory to calculate the communicative success (usage:`calc_average_success window input-file [> output-file]`).

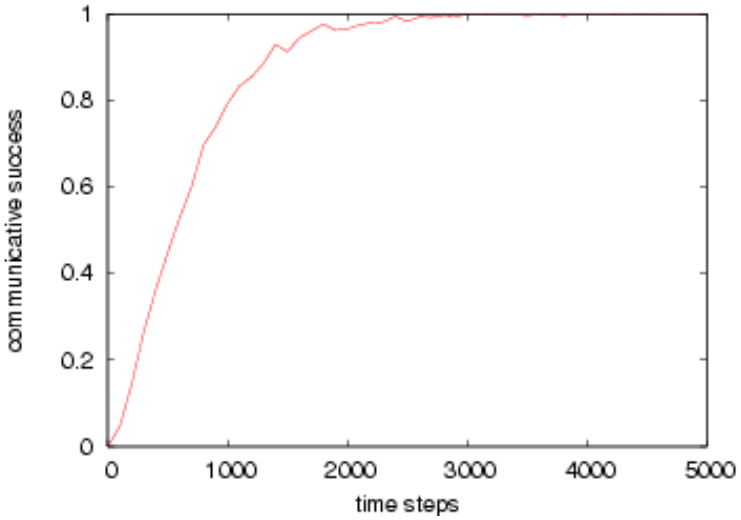

Figure 1: Communicative success for 10 runs of guessing games with 10 agents and 25 meanings.

After implementation, play around with the system and check whether you get a typical curve of *communicative success* as in figure 1. Note that this curve was generated with 10 agents and 25 meanings.

2.2 Implement a discrimination game

In this part, we move towards grounding the language. We will assume a world that contains randomised objects (e.g., coloured shapes as in the Talking Heads experiment), which are created with 3 features indicating colour (e.g., the RGB values) and a 4th feature indicating shape (e.g., the ratio between surface area of a shape divided by its smallest rectangular bounding box, so that squares have value 1, triangles value 0.5, etc).

For the moment, you should generate these the objects by generating random values between 0 and 1 for each feature of each object. I.e., for each object you create four random values to construct the object's feature value. In addition, we shall assume a single agent constructing meanings from these feature values.

Now implement the following procedure:

1. Loop for N discrimination games.
2. Create 4 random objects to form the context.
3. Select one arbitrary object to form the topic of the game.
4. Categorise all objects, using a representation of your choice (e.g., prototype, binary tree, neural network), as long as you can attach a label to it.

5. Verify whether the topic's category is distinctive of not and adapt –when necessary– the *ontology* (the list of categories that will form your meanings).

Before moving on, test the discrimination game by running it for a few 100 games and measure its success (again by printing 0 and 1 for failure or success). If all goes well, discriminative success should rapidly increase to 100. Increasing the context size from 4 to 8 or the number of dimensions should affect the outcome.

2.3 Integrating naming and discrimination games

Now, integrate the naming game with the discrimination by having each game generate a context as with the discrimination games and have each agent generate meanings using the discrimination games. Naturally, the naming game used earlier should be adapted to use these grounded meanings rather than the integers used before.

It would be nice to generate a visualisation of the categories that are generated. It is also interesting to test the difference between randomly generated objects and those generated from a large set of colours taking from natural and/or urban pictures. A data set generated by Belpaeme and Bleys [1] for their study on colour naming is provided in the `data` directory (see http://www.tech.plym.ac.uk/SoCCE/staff/TonyBelpaeme/).

2.4 The research plan

At this point the basic language model has been implemented. Now you should think of a research question and a plan for investigating the that plan. Try to remain modest in the question and plan, since little time remains. Many parameters can be investigated. Some of these include:

- Compare various language learning mechanisms
- Compare various categorisation mechanisms
- Investigate issues of language contact
- Investigate issues of spatial organisation of population
- Investigate issues of social networks
- Investigate issues of transgenerational transmission
- Investigate issues of compositionality
- ...

Adapt your program to allow research into the plan and carry out the research. Also consider writing the report in time.

3 THSim

The Talking Heads simulation toolkit has already the above –and much more– implemented, it has a graphical user interface and many parameters that can be set, so it can be directly used for research. Moreover, Java programmers can adapt the system to their own needs.

The downside of using it is that it is perhaps less flexible and it will take a while before you really understand what is going on inside. Nevertheless, feel free to use it.

The package is found in the `thsim.v4.03` directory and can be started using the command `java THSim` or `runthsim`. The latter is adviced, since it allocates enough heap memory to do interesting things. Do you require more or less memory, you can specify this in the options. Usage:

```
./runthsim [--a java-options --c classpath --d working-dir
   --n noruns --m heap-memory-size --o outfile --h] args...
```

Note heap-memory-size should end with M (or MB), `args` are in the form of `-option value`... Use ./runthsim -h for help on THSim options and values.

For further documentation, first consult the Roughguide.v4.0.pdf in the `docs` directory. The paper /em THSim v3.2: The Talking Heads simulation tool [2] contains a functional description of an earlier version. The compositional agents are described in [3]. More information, pointers to papers etc. are found on http://ilk.uvt.nl/~pvogt.

References

[1] T. Belpaeme and J. Bleys. Explaining universal colour categories through a constrained acquisition process. *Adaptive Behavior*, 2005. This issue.

[2] P. Vogt. THSim v3.2: The Talking Heads simulation tool. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors, *Advances in Artificial Life - Proceedings of the 7th European Conference on Artificial Life (ECAL)*. Springer Verlag Berlin, Heidelberg, 2003.

[3] P. Vogt. The emergence of compositional structures in perceptually grounded language games. *Artificial Intelligence*, 167(1–2):206–242, 2005.