Chapter 1

An Algorithm for Bootstrapping Communications

Jacob Beal
MIT AI Lab
jakebeal@mit.edu

In a distributed model of intelligence, peer components need to communicate with one another. I present a system which enables two agents connected by a thick twisted bundle of wires to bootstrap a simple communication system from observations of a shared environment. The agents learn a large vocabulary of symbols, as well as inflections on those symbols which allow thematic role-frames to be transmitted. Language acquisition time is rapid and linear in the number of symbols and inflections. The final communication system is robust and performance degrades gradually in the face of problems.

1.1 Introduction

Neuroscience has postulated that the brain has many "organs" — internal subdivisions which specialize in one area. If we accept this view, then we need some sort of mechanism to interface these components. The design of this mechanism is limited by the hardware which the brain is constructed out of, as well as the size of the blueprints specifying how it is built. Neurons, as hardware, are relatively slow and imprecise devices, but they are very cheap, and it's easy to throw a lot of them at a problem in parallel. Our DNA is only about 1 gigabyte, too small to encode the full complexity of interfaces between all of the different components.

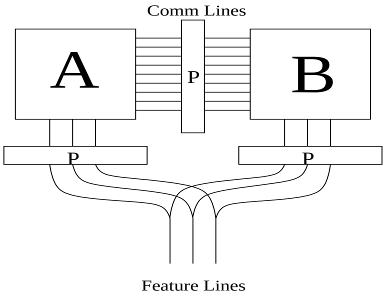


Figure 1.1: The agents labelled A and B are interconnected by *comm lines* — a bundle of wires with an arbitrary and unknown permutation. The agents also share some *feature lines* with the outside world, again with unknown permutations.

I approached this design problem from a hardware hacking point of view, with the question, "If I were designing the human brain, how would I build this interface?" It needs to be self-configuring, to beat the limited blueprints problem, and it needs to learn quickly. On the other hand, hardware is very cheap, and I can design in a domain with a huge number of interface wires between two components.

I have developed an algorithm which bootstraps communications solely from shared experience and I present it here as an existence proof and a tool for thinking about how a brain might be composed out of independent parts that learn to communicate with each other: it is possible for two agents to rapidly construct a language which enables them to communicate robustly.

1.2 System Model

There are two agents in the system, connected by a very large bundle of wires called *comm lines*. Each agent has another set of wires called *feature lines*, over which external information can arrive. Metaphorically, the comm lines are a nerve bundle connecting two regions of the brain and the feature lines are nerve bundles that carry part of the brain's observation of the outside world. The actual wires in the bundles might be arbitrarily twisted and rearranged between the two ends of the system, so we add an unknown permutation to each bundle to model this effect and prevent any implicit sharing of ordering information between the two agents.

The comm lines have four states: 1,-1,0, and X. When undriven, the line reads as 0. Information is sent over the line by driving it to 1 or -1, and if the line is being driven to both 1 and -1, it reads as a conflict — X. In the experiments conducted here, I used a set of 10,000 comm lines.

Feature lines are named with a symbol which they represent and read as undriven, driven, or driven with another symbol. In the experiments I conducted, names of feature lines are things or actions and the symbols driven on them are

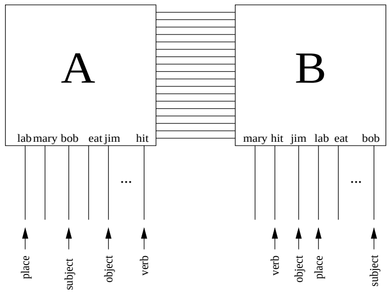


Figure 1.2: During a training cycle, the feature lines of both units are driven. Each agent attempts to learn from the comm lines driven by the other agent.

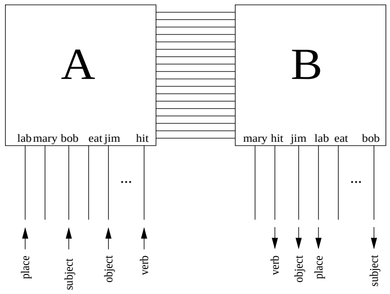


Figure 1.3: During a test cycle, the feature lines of one unit, say A, are driven and the feature lines of the other unit are observed. The test is scored by number of mistakes in B's reproduction of A's feature lines.

roles. So typical feature lines might be **bob**, **mary**, or **push**, and typical roles might be **subject**, **object**, or **verb**. The set of feature lines is of undefined size — the agents have no knowledge of what feature names or roles exist until they encounter them in practice.

An agent can read and drive both comm and feature lines, but are constrained to synchronous schedule. Each cycle of the system has a "talk" phase and a "listen" phase. Agents can read lines at any time, but can only drive comm lines during the talk phase and feature lines during the listen phase. At the end of the talk phase, for symmetry breaking purposes one agent is randomly selected to have spoken first. The agent which spoke first can read the results of both agents speaking in its listen phase, while the one which spoke second reads only what the first agent spoke.

There are two types of cycles — training cycles and test cycles. In a training cycle, the data on the feature lines is sent to both agents. In a test cycle, one agent is randomly selected to receive input from the feature lines, while the other receives no input. Performance may then be evaluated on the basis of how well the output of the agent receiving no input matches the values on the feature lines.

1.3 Algorithm

The key idea driving this algorithm is that sparseness makes it easy to separate the stimuli.

Knowledge in the system is represented by two sets of mappings: symbol mappings and inflection mappings. An inflection mapping links a symbol carried on a feature line to a real value between 0 and 1. A symbol mapping links a feature line with two sets of comm lines, designated as *certain* and *uncertain*, and includes an integer designated *certainty*.

These mappings are used symmetrically for production and interpretation of messages. In the "talk" phase, each driven feature line selects the *certain* comm lines associated via the symbol mapping and drives them with the unary fraction associated with the symbol on the feature line via the inflection mapping. In the "listen" phase, if enough of a feature line's associated comm lines are driven, then the feature line is driven with any inflection mapping symbols within a fixed radius of the unary code on that set of comm lines.

Both types of mappings are generated randomly when a feature or inflection is first encountered, then adjusted based on observations of the other agent's transmissions. These adjustments take place only if an agent spoke second; if it was the first one to speak, then its own transmissions are on the lines as well, inextricably mixed with the transmissions of the second agent, and this would make accurate learning significantly more difficult.

Inflection mappings are adjusted with a very simple agreement algorithm: if the received unary code is significantly different from expected, the code in the mapping is set to the received value. If a unary code matches which should not have, then it is purged and generated anew.

Symbol mappings are slightly more complicated. The first time an agent hears a given symbol spoken by the other agent, it adds every driven comm line to the *uncertain* lines for that symbol. Each time thereafter that it hears the symbol again, it intersects the driven lines with its *uncertain* lines, thereby eliminating lines associated with other symbols. After several iterations of this, it assumes that there is nothing left but lines which should be associated with the symbol, adds the *uncertain* lines to the *certain* lines, and begins to use them for communication. A few more iterations after that, it begins paring down the *certain* lines the same way, so that the two agents can be assured that they have identical mappings for the symbol.

A more detailed description of the algorithm, including code implementing it, may be found in [1] and [2].

1.4 Results

To test the algorithm, I used a system with an n_w of 10000 comm-lines and a n_{wps} of 100 random wires selected to generate a new symbol mapping.

I trained the system for 1000 cycles, then evaluated its performance over an additional 200 cycles. Each cycle, an example is generated and presented to the system. In the training phase, there is an 80% chance it will be presented to both agents and a 20% chance it will be presented to only one (That is, 80% training cycles, 20% test cycles). During the evaluation phase, the first 100 are presented to the first agent only, and the second 100 are presented to the second agent only. A test is considered successful if the input feature set is exactly reproduced by the listening agent.

The examples input to the feature lines are thematic role frames generated from a set of 50 nouns, 20 verbs, and 4 noun-roles. Each example is randomly generated with 0-2 verbs assigned the "verb" role and 2-4 nouns assigned noun-roles. No noun, verb, or noun-role can appear more than once in an example. A typical scene, then, might be `'((approach verb) (jim subject) (shovel instrument) (lab object))`, which corresponds loosely to "Jim approached the lab with the shovel." All told, there are more than 1.2 billion examples which can be generated by the system, so in general an agent will never see a given scene twice.

In a typical run of this system, after about 200 cycles most symbols will have entered the shared vocabulary and can be successfully communicated between the two agents. After about 500 cycles, the set of inflections will have stabilized as well. In the final round of 200 tests, the success rate is usually 100%, although occasionally due to the stochastic nature of the algorithm, the inflections will not yet have converged by the end of 1000 tests and consequently one or more will not be transmitted correctly.

1.4.1 Convergence Time

The time needed to develop a shared vocabulary is proportional to the number of symbols in the vocabulary. A symbol is learned when both agents have *certainty* for that symbol greater than t_c. An agent increases *certainty* when it speaks second, which is determined randomly, so we may estimate this as a Poisson process. Thus, we may calculate the expected number of cycles, c, as follows:

$$E(c) = 2t_c \frac{\binom{2t_c}{t_c}}{2^{2t_c}} + \sum_{n=2t_c+1}^{\infty} n \frac{\binom{n-1}{t_c-1}}{2^{n-1}}$$

Evaluating this for $t_c = 4$, we find an expectation of 10.187 uses of a symbol before both *certainty* thresholds are reached.

For these experiments then, with an average of 3 nouns and 1 verb per training cycle, then, we can calculate the expected number of shared symbols S as a function of elapsed cycles t:

$$S(t) = n_{nouns} * (1 - P(10.187, t\frac{3}{n_{nouns}}0.8)) + n_{verbs} * (1 - P(10.187, t\frac{1}{n_{verbs}}0.8))$$

where P is the incomplete gamma function. Since this function is linear in the number of symbols, we see that the time to build a shared vocabulary is

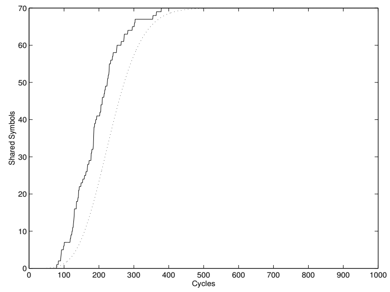


Figure 1.4: Number of shared symbols versus elapsed time for 50 nouns and 20 verbs. Dotted line is theoretical estimate S(t), solid line is experimental data.

linear in the number of symbols. Figure 1.4 shows experimental data confirming this estimate.

Once a shared vocabulary of symbols exists, the algorithm can begin learning inflections. If n_i is the number of inflections to be learned, and r_i is chosen such that $r_i * n_i \leq 0.5$, then we can show that the time to develop a shared set of inflections is $O(n_i)$.

An inflection may be learned any time a symbol is successfully transmitted in a training cycle. This occurs if the new inflection does not conflict with any of the previously learned inflections - that is, if n symbols have already been learned, then it must be the case that for all v_i s.t. $1 \leq i \leq n$, $|v_{n+1} - v_i| < 2r_i$. Since the value of the new symbol, v_{n+1}, is chosen by a uniform random process on the interval $[0, 1]$, the probability p_{n+1} of choosing an acceptable inflection value is no less than $1 - (2r_i * n)$. The n_ith inflection, then, has the least probability of success, $p_{n_i} = 1 - (2r_i * (n_i - 1)) \geq 2r_i$, and p_n is generally bounded below by $2r_i$.

For these experiments then, we can calculate the expected number of inflections, assuming a shared vocabulary, as a function $I(t)$ of elapsed cycles t. There are expected to be 3 noun inflections and 1 verb inflection per training cycle, so the least frequent inflection is expected to appear at with frequency at least $1/n_i$. Thus, we obtain

$$I(t) = n_i * (1 - P(1, 2r_i t \frac{1}{n_i} 0.8))$$

where P is the incomplete gamma function. Since this function is linear in the number of inflections, we see that the time to build a shared set of inflections is linear in the number of inflections. Figure 1.5 shows experimental data confirming this estimate.

Thus, the algorithm is expected to converge in $O(s + n_i)$ time, where s is the size of the vocabulary and n_i is the number of inflections.

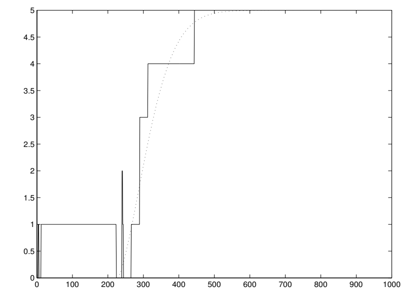


Figure 1.5: Number of shared inflections versus elapsed time for 4 noun inflections and 1 verb inflection, in a system with 50 nouns and 20 verbs. The dotted line is theoretical estimate I(t), beginning with cycle 230, where S(t) predicts half the vocabulary to be learned. The solid line is experimental data.

1.4.2 Channel Capacity

The number of symbols and roles which can be learned without false symbol detection and inflection misinterpretation is dependent on the number of wires n_w, the number of wires per symbol n_{wps}, and the percent stimulus necessary to recognize a symbol p_s.

If we want no combination of symbols to be able to generate a spurious recognition, then each symbol must have at least $n_{wps}(1-p_s)$ wires not used by any other symbol. This means that a vocabulary would have a maximum size of only $\frac{n_w}{n_{wps}(1-p_s)}$. In practice, however, we can assume that only a few symbols are being transmitted simultaneously. If we assume that no more than m symbols will be transmitted at once, then we can conservatively estimate capacity by allowing any two symbols to overlap by no more than $n_{wps} * p_s/m$ wires. Thus any given symbol covers a portion of symbol space with volume:

$$\sum_{i=0}^{n_{wps}(1-\frac{p_s}{m})} \binom{n_{wps}}{i} \binom{n_w - n_{wps}}{i}$$

The whole symbol space has volume $\binom{n_w}{n_{wps}}$, so a conservative estimate of the maximum number of symbols that can exist is:

$$\frac{\binom{n_w}{n_{wps}}}{\sum_{i=0}^{n_{wps}(1-\frac{p_s}{m})} \binom{n_{wps}}{i} \binom{n_w - n_{wps}}{i}}$$

This yields a satisfactorily large capacity for symbols. For the experiments described above, with $n_w = 10000$, $n_{wps} = 100$, $p_s = 0.8$ and a maximum of 6 concurrent symbols, we find that the capacity is 1.167×10^{12} distinct symbols.

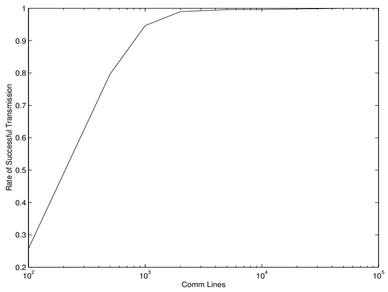


Figure 1.6: Number of comm lines versus transmission robustness. Horizontal axis is n_w from 100 to 100,000. Vertical axis shows the symbols and inflections correctly received per symbol and inflection transmitted (spurious receptions count against this as well) over the course of 200 test cycles on a system trained with 50 nouns, 20 verbs and 4 noun-roles, $n_{wps} = 20$, $p_s = 0.8$. Accuracy degrades smoothly with decreased channel capacity.

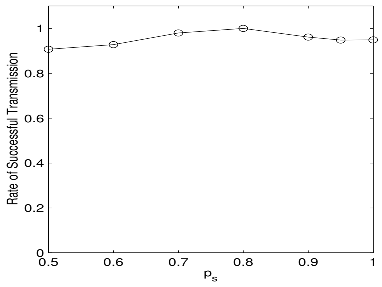
1.4.3 Performance Degradation

We expect that the performance of the algorithm will degrade gracefully as the channel capacity is reduced. As the average Hamming distance between symbols drops, the chance that a combination of other symbols will overlap to produce a spurious recognition or interfere with the inflection being transmitted rises. Since symbols receiving too much interference are discarded, the algorithm will tend to break up clusters of symbols and move toward an efficient filling of symbol space. Thus, reducing the ratio n_w/n_{wps} ought to cause the transmission errors to rise gradually and smoothly. In practice we find that this is in fact the case, as shown in Figure 1.6.

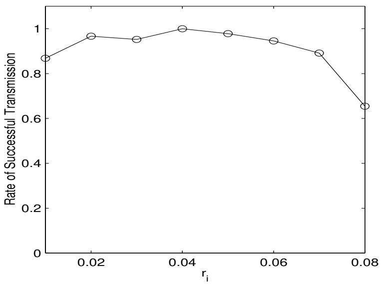
1.4.4 Parameter Variation

The values of the parameters used in the experiments above were not carefully chosen. Rather, I made a guess at a reasonable value for each parameter, expecting that the algorithm should not be very sensitive to the parameter values. (If it were, then I could hardly claim it was a robust algorithm!)

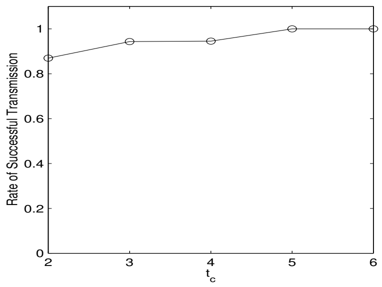
To test this, I ran a series of experiments in which I trained and tested the system with one of the parameters set to a different value. For each value for each parameter I ran 10 experiments: Figure 1.7 shows the performance of the algorithm as a function of parameter value for each of the six parameters p_s, r_i, t_c, t_p, w_m, and n_{wps}. (n_w is excluded because its variation is evaluated in the preceding section) As predicted, the performance of the algorithm is good over a wide range of values for each variable.



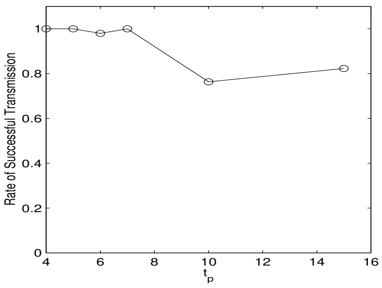
(a) Variation of p_s



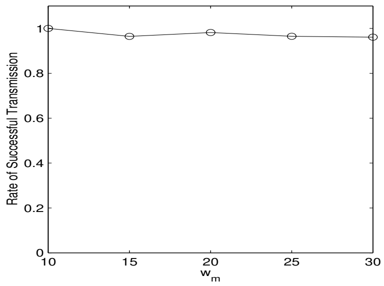
(b) Variation of r_i



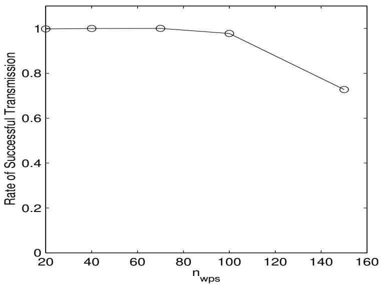
(c) Variation of t_c



(d) Variation of t_p



(e) Variation of w_m



(f) Variation of n_{wps}

Figure 1.7: Variation in performance as each parameter is varied. For each graph, the horizontal axis shows the value of the parameter being varied and the vertical axis shows the fraction of symbols and inflections correctly received per symbol and inflection transmitted. Measurements are the average values over the course of 10 runs of 200 test cycles, as in 1.6. For each run of test cycles, the systems were trained with 50 nouns, 20 verbs and 4 noun-roles, with base parameter values $p_s = 0.8$, $r_i = 0.05$, $t_c = 4$, $t_p = 6$, $w_m = 20$, $n_{wps} = 100$, and $n_w = 10000$. All parameters in the system can tolerate small variations without serious degradation in performance.

1.5 Contributions

I have built an algorithm which allows two agents to generate a shared language on the basis of shared experiences only. The behavior of this algorithm can be analyzed and performs as predicted by theoretical analysis.

1.6 Acknowledgements

Particular note should be given to the help from several people. Gerry Sussman started me thinking about the problem and pointed me in this direction. My research is part of a continuing effort started by Yip and Sussman to build a "TTL databook for the mind" — a compatible system of modules that capture aspects of mental activity and can be combined to build ever more powerful systems. Thanks also to Catherine Havasi for hacking some of the early code with me and being a needling presence to keep me from slacking off.

Bibliography

[1] BEAL, Jacob. "An Algorithm for Bootstrapping Communications" MIT AI Memo 2001-016, August, 2001.

[2] BEAL, Jacob. "Generating Communications Systems Through Shared Context" MIT AI Technical Report 2002-002, January, 2002.

[3] KIRBY, Simon. "Language evolution without natural selection: From vocabulary to syntax in a population of learners." Edinburgh Occasional Paper in Linguistics EOPL-98-1, 1998. University of Edinburgh Department of Linguistics.

[4] KIRBY, Simon. "Learning, Bottlenecks and the Evolution of Recursive Syntax." Linguistic Evolution through Language Acquisition: Formal and Computational Models edited by Ted Briscoe. Cambridge University Press, in Press.

[5] MINSKY, Marvin. *The Society of Mind.* Simon & Schuster, Inc, New York, 1985.

[6] MOOERS, Calvin. "Putting Probability to Work in Coding Punched Cards: Zatocoding (Zator Technical Bulletin No. 10), 1947. Reprinted as Zator Technical Bulletin No. 65 (1951).

[7] YIP, Kenneth and SUSSMAN, Gerald Jay. "Sparse Representations for Fast, One-Shot Learning." MIT AI Lab Memo 1633, May 1998.