Simms AI v1 : On the Design of a Self-Trained Conversational Artificial Neural Network

Christopher Mitchell

Abstract

The computational power of current technology is rapidly approaching the capacity of the human brain, yet this power must be reliably harnessed to create artificial pseudointelligences to interact with humans. The vast scope of human thought and conversation make manually programming all possible functions into such an AI infeasible, demonstrating the need for a partially self-trained AI. Simms AI v1 is such an implementation, interacting with humans via instant messaging and reading the open-source encyclopedia Wikipedia to gain a knowledge of English syntax and vocabulary and build its speech patterns. It uses a context-based algorithm that performed well under controlled circumstances, and showed the flaws that must be improved in the next version. Simms v1 demonstrates that a complex neural network can be self-trained to parse and construct syntactically-correct sentences without any a priori knowledge of a language’s syntax and vocabulary.

1 Introduction

Simms AI v1 is the culmination of several previous Cemetech projects including WordNet, an application that attempted to spider websites and build associations between related websites based on keywords rather than mutual hyperlinking. The basic design of Simms is a centralized neural network database paired with a distributed processing array that parses input into long-term memory, constructs a response when necessary, and returns it to the module that delivered the input. Its neural structure is based on that of the human brain, in that each neuron or ’node’ maintains between one and thousands of connections to other nodes. In order to speed the development of its associative memory, a variety of techniques were employed, including spidering Wikipedia, reading chat logs, and engaging in conversation with human users, which quickly demonstrated the power and the limitations of the approach. Speed and working database size became a primary concern as Simms’ neural network exceeded 300,000 nodes connected by two million connections, a suboptimal node-connection ratio caused by poor text-cleaning routines and storage efficiency. Careful analysis revealed fundamental flaws in the neural storage system that will be examined in designing Simms AI v2, the next iteration of the Simms project.

Simms’ software was written from scratch using command-line (CLI) PHP and MySQL. A custom thread-handling system was designed and constructed that can manage multiple modules, managers, and database accesses over many physical machines. Simms AI v1 comprises a single supervisor node containing the neural network database, and six slave nodes performing computation and interaction. Each node is assigned a specified role, which sometimes leads to misbalanced load; Simms AI v2 will incorporate automatic load balancing to distribute work

CEMETECH
Cemetech Technical Report #2
(c) 2009 Christopher Mitchell
March 30, 2009

more evenly. Each node also runs a manager thread that is responsible for notifying the supervisor that it is still active and receiving new versions of modules to allow for on-the-fly, zero-downtime upgrades.

A primary focus in the design of Simms AI v1 was a lack of reliance on any pre-programmed syntactical knowledge of English specifically other than its parsing system's delineation of phrases at space and punctuation boundaries. In theory, if trained primarily in any one language, it would begin to speak and respond in that language using vocabulary and syntax appropriate to the language. Simms AI v1 does not have the ability to associate similar forms of the same word and extract rules for conjugation or declination, instead assuming that it sees each form of a word in the proper context with sufficient frequency to train it in the correct usage of each form. Once Simms AI had read a large number of articles and held conversations over several months, it was indeed capable of parsing long sentences and returning similarly syntactically-valid English sentences with correct vocabulary and tense and number agreement, but continues to lack any understanding of the sentences it is constructing, obviating the need for a second version building on the successes of Simms AI v1 and correcting its inadequencies.

2 Neural and Module Structure

Simms AI consists of four layers of data storage and processing modules. The bottom of the hierarchy illustrated in Figure 1 is the Long-Term Memory (LTM) database, the primary purpose of which is to store the two massive tables that form Simms AI's merged syntactical and factual knowledge. It also holds management tables recording the active threads across all clustered machines and managing inter-process signaling, which is abstracted such that the differences between inter- and intra-node communications are transparent to the individual threads. The next highest layer is the logic algorithm that controls two-way communication between the top layer (I/O and administration modules) and the database. In one direction it parses an English sentence and stores it into the LTM along with its context; in the opposite direction, it constructs an English sentence for output. LTM is read and written by a variety of modules in the second-to-top layer comprising two main classes, I/O modules and administration modules. The I/O modules take input from protocols such as HTTP and AIM (AOL Instant Messenger) and pass it down to the Brain level, then wait for a response from the Brain level and return it to the originating protocol. In addition, some handle multiple sessions, like tracking different contexts for conversations with different users on AIM. The top layer is a management system that attempts to maximize uptime by killing lagging threads, respawning dead threads, moderating ITC (Inter-Thread Communication), and synchronizing updated modules across all cluster nodes.

2.1 Memory Structure

Simms' long-term memory (LTM) is designed to simultaneously store factual and syntactical information, a choice with advantages and disadvantages. Combining the two sets of data simplifies LTM reads and writes and therefore makes them faster, but also significantly limits the variety and coherence of the responses that Simms 1 can produce. The LTM is stored as a vast MySQL database with two primary tables, Nodes and Connections. The items in the Nodes table each represent a single English word, number, or other alphanumeric sequence that can be inserted into a sentence with a delimiter such as space, comma, or period on each side. Each word is linked to a unique identifying number, and also has associated metrics tracking its hitcount (the number of connections it holds to other Nodes) and timestamps for its creation and last access. The Connections table tracks all of the possible directional paths from one node to another; each item in Connections has a from and to ID number corresponding to items in the Nodes table, a strength value that increases as each connection is reinforced by usage or

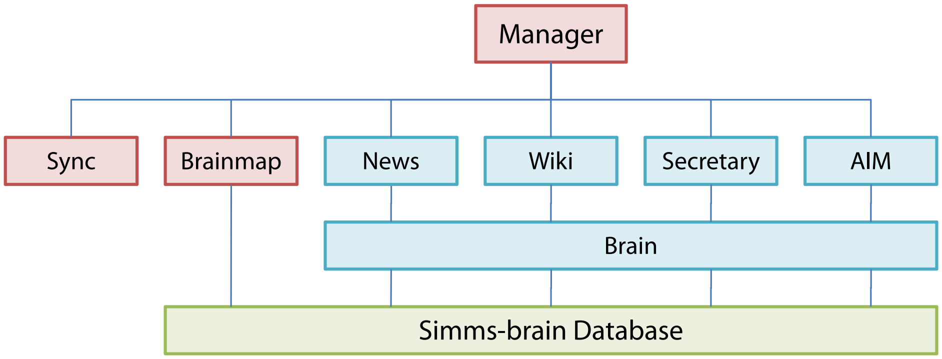


Fig. 1. In each hardware node, the manager module governs, spawns, and kills each of the other modules, including the Sync and Brainmap management modules and the Wiki, Secretary, and AIM interaction modules. The interaction modules use the Brain module to insert and extract coherent conversational data from the main neural database of Simms' long-term memory. Short-term memory is currently stored in temporary arrays within the individual interaction processes

learning, and a list of associated contexts or topics, also a set of integer identifiers referring to the Nodes table. While powerful enough to allow the construction of novel, grammatically-correct sentences, this system proved infeasible for the storage and retrieval of contextually-relevant information. Specifically, Simms 1 has trouble figuring out which of the words or phrases in a sentence are the most important to use to set the context, and while cross-indexing the relative strengths of each of the possible keywords would slightly improve this ability, the computation cost would be too high to be practical.

A practical example of the structure of Simms' neural network is shown in Figure 2. Each of the rectangles are items in the Nodes table with their English equivalents. The arrows connecting nodes represent items in the Connections tables; note that connections are unidirectional. This diagram is centered on the structure created by the sentence "Simms AI is a machine."; lighter arrows show the connections formed by the sentence "What is Simms AI?" Finally, an additional unlabeled branch shows how the sentence "Simms AI is a robot" branches off of the "Simms AI is a machine" connection structure. Notice that each connection is tagged with the topics it involves, specifically, all the words in the current context (for a longer conversation, this context would include words from previous sentences). Note how Simms is able to construct a new sentences from the previous valid connections, starting at the "What" node and working through "is a" to finish with either "machine" or "robot", creating "What is a robot?" or "What is a machine?" The disadvantage of this system is the valid construction of nonsensical sentences like "What is Simms AI is a robot?".

2.2 AIM Module

The AIM module governs Simms AI's interactions with users over the AOL Instant Messenger Protocol used by a large percentage of instant messaging clients currently available. It utilizes the BlueTOC PHP library to communicate with the Internet, separating conversations with each user into a subthread to avoid excessive lag caused by high database usage. Simms treats each simultaneous conversation separately with a separate context, though in the future conferences may be possible that share contexts. The AIM module logs all conversations for later examination.

2.3 Wikipedia Module

Wikipedia is an example of a large user-created database written in a combination of technical and conversational English. Simms reserves a thread to select random topics from the Ency-

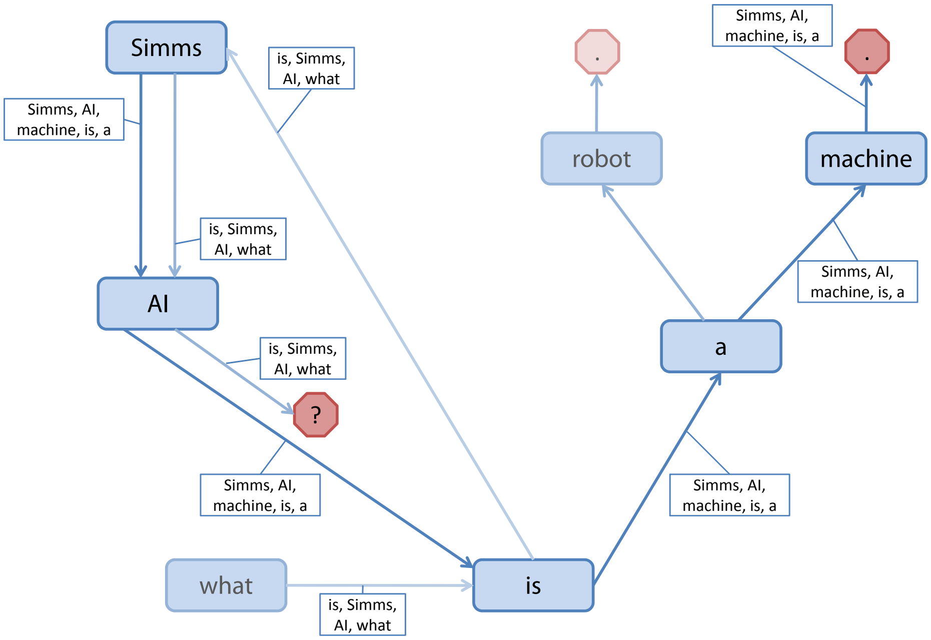

Simms
is, Simms, AI, what
Simms, AI, machine, is, a
is, Simms, AI, what
AI
is, Simms, AI, what
?
Simms, AI, machine, is, a
what
is, Simms, AI, what
is
.
Simms, AI, machine, is, a
.
robot
machine
Simms, AI, machine, is, a
a
Simms, AI, machine, is, a

Fig. 2. This diagram shows the partial neural network for the phrases ”What is Simms AI?”, ”Simms AI is a machine”, and ”Simms AI is a robot”. It demonstrates other phrases that Simms might be able to synthesize, such as ”What is a machine?” and ”What is a robot?”.

clopedia to research both to build its syntactical space and to increase its factual associations. The Wikipedia module picks a random word from the Nodes table, attempts to look it up, and repeats until it finds an article. It properly parses redirects within the MediaWiki markup, and parses each page it reads to remove as much formatting as possible to render the page down to plain text. Wikipedia poses a problem for conversational learning, as it is written from a single perspective. Future implementations will use Wikipedia and similar databases to build syntactical and factual knowledge but will maintain a separate contextual system to determine how input from another user should be handled and how a relevant response should be framed. Once Simms finishes parsing each sentence, maintaining a continuous context through the article, it flushes the context and starts looking for a new article.

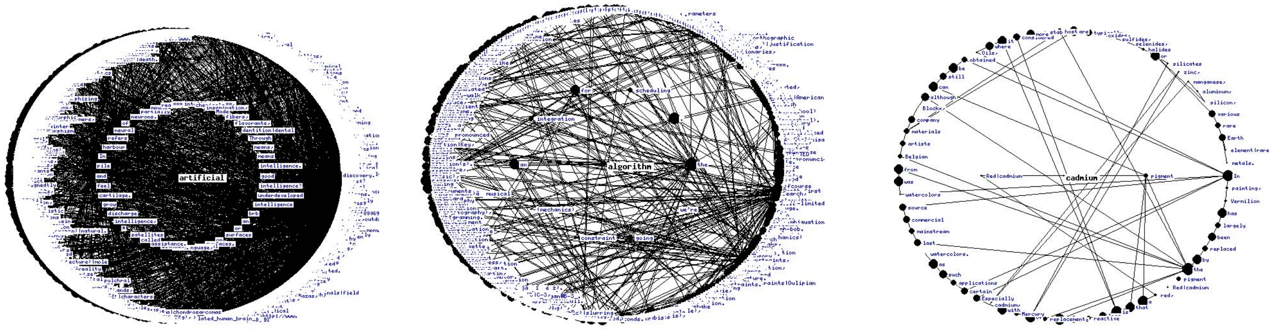
2.4 Secretary Module

The Secretary module serves the dual purpose of logging attempts to reach a real human user when that user is offline, and takes advantage of normal people to build Simms’ conversational experience. The AIM module monitors the screenname of Simms’ creator and opens a second AIM session under that username when the human is detected to have logged off. It logs any conversations that occur between the human and other human users, and also attempts to interact by introducing itself and then responding to the other human users.

2.5 Brainmap Module

In order to better visualize the contents of Simms’ memory and its association mechanism, the Brainmap module accepts a keyword and generates an image of two levels of neural network from that particular keyword. The generated neural map centered on the keyword with the top m nodes associated with the keyword arranged in a circle around it and an outer ring of

n nodes connected to the first-level nodes. In addition, the Brainmap module can accept web-based requests and email the final image to a user when it completes. The Brainmap module is particularly helpful in visualizing for what topics Simms AI has much or little context, as shown in Figure 3.



(a) High context density (b) Moderate context density (c) Low context density

Fig. 3. Brainmaps generated for three topics showing varying context density, 'artificial', 'algorithm', and 'cadmium' respectively.

2.6 Other Modules

Many other modules and smaller blocks of code maintain Simms AI's structure and functionality. Among the most important are the Sync module, the News module, and the TTY handler. Sync is an administrative module that rsyncs from the supervisor node machine (Simms) to the subsidiary processing nodes (Simms0 - Simms5). The News module is an I/O module similar in function to the Wikipedia module; it reads the RSS feed for Google News every five minutes, parses it into headlines, and uses the Brain module to parse each headline into its database, thus keeping its vocabulary up-to-date with current events. The TTY handler is a sub-module within the Manager that displays a running status display on the LCD of the supervisor node of thread births, deaths, and suicides, important thread events, and error/debug messages.

3 Clustered Hardware Implementation

Simms runs on a custom PHP cluster management system that transparently executes and synchronizes threads across multiple physical node. One machine is designated as the supervisor machine (in the instance of Simms, as $Simms$). All the other machines in the cluster are normal processing nodes (labeled $Simms0$ through $Simms5$ for this case). Rather than dynamically allocate available processing power, each of Simms' six non-supervisor nodes is assigned a specific role: Simms0 handles the AIM and Secretary modules, Simms1 runs the Wiki module, Simms2 maintains the Subconscious module (a "creativity" system that was never completed in Simms AI v1 but will be implemented in some form in Simms AI v2), Simm3 runs three parallel Brainmap threads, Simms4 handles on-the-fly LTM backups, and Simms5 has no assigned role.

The thread management system uses the supervisor node as the scheduler, if Simms' threading system was likened to a single-computer task scheduler, and each of the machines can be taken as a multithreaded CPU. The supervisor ensures that a Manager module is running on each of the nodes (including itself), and each Manager is in charge of maintaining the correct threads on its own node. The Managers have the power to spawn new threads, kill existing threads that are lagged, and accept the suicide of a single-execution thread that has completed its duties. Any thread can send signals to any other thread, as moderated through the Simms-brain database my the Manager on the supervisor node, with one specific limitation: signals can only name recipient and sender of a signal as a thread of a specific type (Wiki, AIM, Brainmap, etc), not a particular instance of that thread type. Therefore, signals must pertain to stateless transactions and can be fetched and cleared by the first thread of the target type that is available, a mechanism that greatly reduces lag when a thread of one type is loaded much more heavily than another thread of the same type.

Only the supervisor node ($Simms$) operates with a monitor to display TTY output of errors and events; the rest operate in headless mode. All are running Ubuntu 7.04 server edition with PHP 5 and MySQL 5. Each of the non-supervisor nodes is a 2.00GHz Pentium 4 (32-bit, 256KB L2 cache) with 1GB RAM and a 20GB or 30GB hard drive. The supervisor is a hyperthreaded Pentium 4 at 3.20GHz (32-bit, 2MB L2 cache total) with 2GB RAM and a 320GB hard drive for the Simms-brain database.

4 Performance

Simms AI was optimized for ease-of-modification rather than pure speed, hence its implementation in PHP and MySQL instead of a faster C or C++ application. The MySQL manipulation was particularly efficient from PHP, and the dynamic variable allocation allowed easy creation and destruction of the large arrays Simms allocates for sentence parsing and synthesis. However, this simplicity introduced a cost in the form of relatively high per-word processing time for both parsing and synthesis, a penalty made particularly acute by the single database used for both syntax/vocabulary and factual information. As expected, storage used for inter-node connections increased linearly given constant uptime (or polynomially given non-constant uptime), while the rate of storage needed for new nodes decreased linearly over time as Simms' vocabulary grew and the discovery of new words became less frequent. Memory and storage usage is discussed in detail and detailed for the first five months of Simms' life in Figure 4.

Processing performance was erratic at the beginning of Simms' testing, but once a few coding errors were found and remedied, a simple and logical pattern was found governing processing time. Longer conversations generally caused longer response time due to a buildup of context, although context decay (10% per sentence) prevented infinite context accumulation. The length of the input sentence was found to be the largest factor in deciding the final response time, while the length of the output sentence synthesized by Simms was relatively uncorrelated to the response time; a mathematical analysis is below along with a summary graphed in Figure 5.

4.1 Memory Growth and Scalability

Simms AI's memory structure necessitates that over time, the delta node count per unit time should decrease, as the longer Simms learns the fewer new words it will run across, a language like English having a finite number of words and word-forms. Since the number of ways that words can be arranged with respect to each other is at most $N_{connections} = 2N_{nodes}^2$, and Simms reads sentences such that for each sentence, in which n_{nodes} is the number of words in the sentence (non-unique) and $n_{connections} = n_{words} - 1$, it must be true that $0.5n_{nodes} \leq n_{connections} < n_{nodes}$, connection growth continues nearly linearly with respect to uptime for a diverse set of inputs even as the growth rate of the nodes table drops off precipitously. This trend is clearly shown in Figure 4, which unfortunately introduces the additional complication of downtime skewing the absolute trends. Nevertheless, the accuracy of the trends for $N_{connections}$ vs N_{nodes} for any time period remains, showing continued linear growth of connections in the database as node count begins to level off.

It is expected that any future implementation would show a similar growth pattern, in which fewer new unique nodes would be discovered per unit time but growth of connections between nodes would remain at least linear. Indeed, given a sufficiently advanced algorithm, connection growth might be expected to be polynomial, if the learning rate accelerated as the implementation became more experienced.

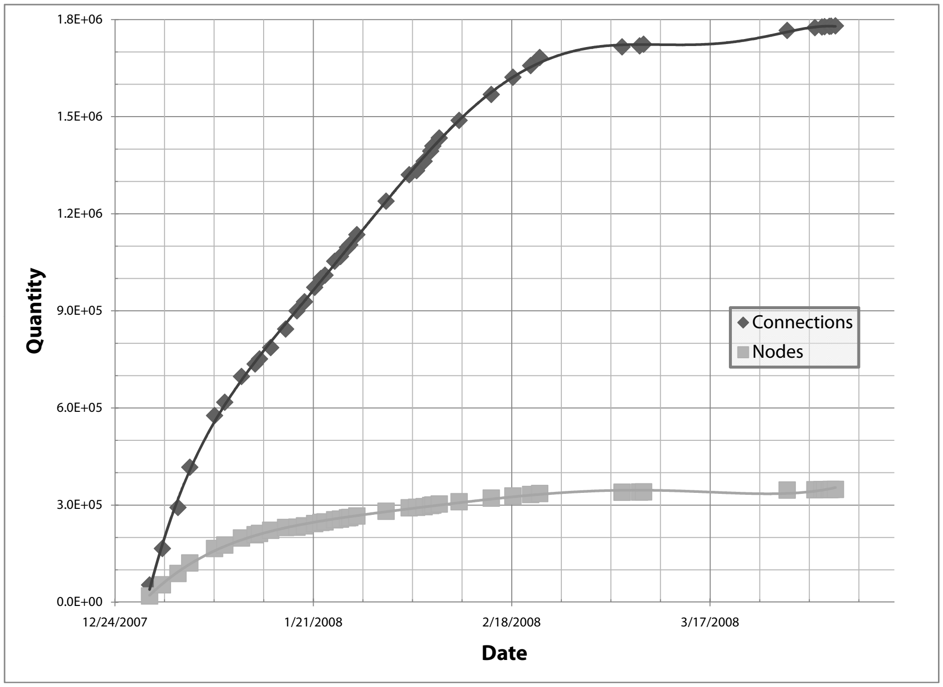
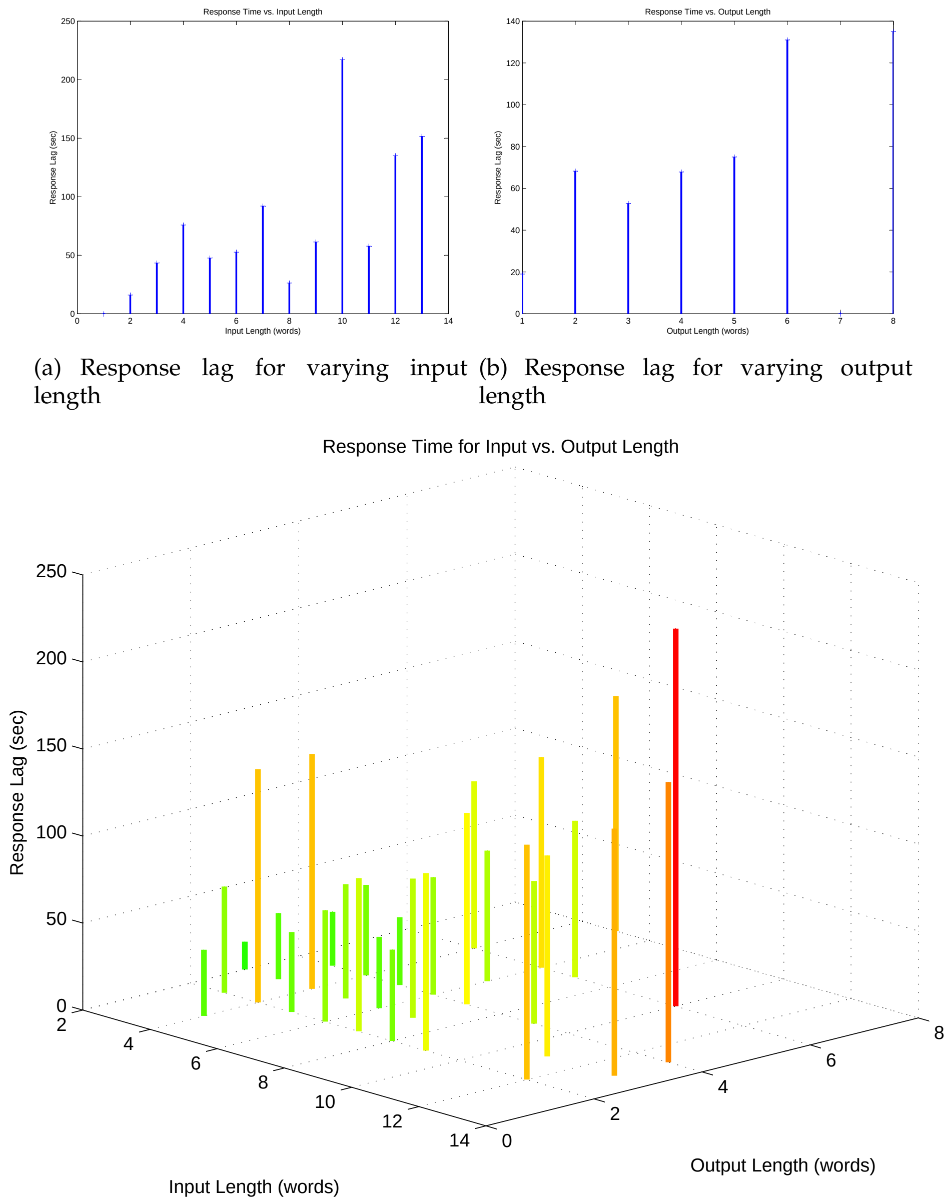


Fig. 4. The growth of Simms' Long-Term Memory (LTM) as it learned and evolved. The notable slowdown in the latter half of the graph is attributable to downtime.

4.2 Processing Performance

Developing a suitable clustered processing implementation proved difficult. At first, the design entailed both a distributed database and distributed processing, but it soon became evident that the network overhead for the whole-table MySQL queries that Simms AI v1 necessitates far outweighed the distributed processing capabilities of such a system. Therefore, in the finalized Simms v1 a single node, $Simms$, takes care of storing the database and serving all query results; actual processing is distributed across each of the other nodes, $Simm0$ through $Simms5$. The lag between an input sentence being delivered to the brain module and the output sentence being returned to the calling module was examined under a variety of parameter variation on the assumption that this most accurately represents the processing performance of Simms. Figure 5 shows three graphs demonstrating response lag as a function of the input sentence length and the output sentence length. As noted below, the nonlinearities and inconsistencies shown by the graphs arise from the effect of context on the brain module. As a conversation grows in duration, context accumulates that makes response more coherent but also lengthens the processing time necessary to return the output sentence. In addition, each unique word has a different amount of related context, and the response time is proportional to the context attached to each word in the input sentence (and to some degree, the output sentence). The most defined trend visible, in Figure 5(a), is that the length of the input sentence bears a significant proportional relationship on the time lag between input and output. The lag also generally increases with output sentence length, but the trend correlation is much less defined than for the input length.

The absolute processing time between input and output is much higher than would be hoped for a conversational neural network; even for short output sentences, response time usually



(a) Response lag for varying input length

(b) Response lag for varying output length

(c) Response lag for varying input/output length

Fig. 5. Response lag in seconds for inputs and outputs of varying lengths. Inconsistencies arise from the different context complexities for each set of input and output; in addition, longer conversations accumulate more context, slowing sentence processing and synthesis.

exceeds one minute. Future versions of Simms will attack this lag in a variety of ways, most importantly the separation of context (factual) and syntactical information to allow fact and syntax to be fetched and combined simultaneously and with both fewer and smaller database queries.

5 Future Work

Simms AI is a thorough proof-of-concept for a conversational, auto-learning neural network pseudointelligence, but its shortcomings and inefficiencies point to many improvements that can be introduced in future versions. One aspect that will not be changed is the implementation in

PHP and MySQL; the ease of modification (PHP is an interpreted rather than compiled language) and seamless integration with the MySQL database query language make it an appropriate choice for implementing a neural-networked AI. As examined in the previous sections, Simms' largest speed handicap derives from its use of a single large database for syntactical, vocabulary, and factual information, a choice that also severely limited its coherence in conversations even with an effort to maintain an extensive context-tracking system for all brain nodes and interactions.

Therefore the primary improvement that will be made in Simms AI v2 is the introduction of separate tables, or possibly even databases, to handle syntactic/vocabulary and factual data. To facilitate self-guided learning, Simms will actively run statistical analysis algorithms against its syntax knowledge in an attempt to find common sentence and phrase structures, group together common types of words (analogous to nouns, verbs, articles, etc, although Simms will not be given an explicit understanding of those classes). Other statistical analysis tools within this ”Subconscious” module grouping may include synonym/antonym searching, word form analysis, and even letter proportion per word analysis, all intended to build up a more intuitive and flexible understanding of the language in question. This improvement will allow a more dynamic response pattern; currently, gregariousness forces Simms to return exactly one sentence for each input sentence. Better-structured factual memory would allow Simms to respond with zero or more sentences to each input sentence, or ideally, even be able to generate spontaneous output with no specific user input.

The database stores significant node and connection noise that clogs the databases, lags queries, and makes responses less accurate. For example, the Wiki module tends to clean input incompletely, occasionally storing markup rather than pure data in Simms' LTM. Better cleaning and demarcation algorithms would improve this efficiency. A secondary improvement will cover both better detection and storage of punctuation within sentences, allowing Simms to understand and use punctuation more correctly.

A final improvement will be made in the clustering algorithm, first to load-balance by automatically assigning new threads to the node within the cluster with the least load, later to subdivide individual threads across multiple nodes to permit sub-thread-level parallelism.

6 Conclusion

Simms AI v1 was shown to be able to synthesize grammatically-consistent sentences given a cohesive training set. Brainmaps revealed that it successfully constructed a neural network where minimum path length between each node is proportional to the relative relevance between each word pair. Its conversational algorithm was less successful; because its responses are based on topic association built from the words the user has typed, it tends to construct sentences containing the highest-scoring word in the input sentence rather than a response with unique words. Future implementations of the Simms project will improve this inadequacy by distinguishing between sentences uttered by different speakers, as well as making a distinction between short-term and long-term memory and between factual and grammatical memory. The cluster-based hardware setup will be maintained and upgraded, with an automatic load-balancing ability built-in to prevent disproportionate load on single hardware nodes.

7 Acknowledgments

The author would like to thank the members of Cemetech for their feedback and stress-testing assistance, Chrystina Montuori Sorrentino and Deian Stefan for their algorithmic advice, and Prof. F. Fontaine and the μLab research laboratory of the Cooper Union for providing the computational resources.