Modeling and Visualizing Dynamic Associative Networks: Towards Developing a More Robust and Biologically-Plausible Cognitive Model

By Michael Zlatkovsky
University of Evansville

ABSTRACT

When modeling imprecise datasets – datasets such as visual sensory input or various other data from the "real world" – traditional computational models are often inadequate at capturing and predicting the data. Various "untraditional" models, such as Artificial Neural Networks, are therefore used for pattern-matching and fuzzy-logic types of tasks. Yet, even despite their greater flexibility over traditional PC-like computational models, ANNs are still limited by their fundamentally static nature, whereby their learning is based on the systematically-assigned weighted connection across the entire network, rather than on the basis and structure of the networks' experiences. Training and re-training the networks therefore requires a tedious and computationally-intensive process of adjusting the networks' weighted connections, until a satisfactory number of inputs produce their corresponding outputs.

My senior project in Modeling and Visualizing Dynamic Associative Networks, under the guidance of Dr. Anthony Beavers, is an attempt to step outside the bounds of this traditional cognitive computational model and study the individual relationship between nodes in a network. Because of their dynamic nature, DANs exhibit ANN-like pattern-matching and rudimentary cognitive abilities, yet they are able to learn continuously (without re-training) and are structured in a biologically-plausible way that directly reflects their underlying knowledge representations.

BACKGROUND: WHY NEURAL NETS, AND WHY *NOT* NEURAL NETS

The desktop PC, despite its raw computational powers, is ill-equipped for pattern-matching, associative tasks, or for dealing with imprecise data (such as visual information obtained the real world, where objects appear in countless variations, occlude each other, or grow unrecognizable under the influence of lighting and shadowing). One alternative to these failings of traditional computational models has been to employ Artificial Neural Networks (ANNs, or "Neural Nets" for short) for these pattern-matching and fuzzy-logic types of tasks. Neural Nets encompasses a fairly broad class of similarly-structured cognitive models, but the variations come more in terms of training the networks, rather than the underlying structure. The networks are composed of a set of "input nodes", a larger and more complex set of "hidden nodes", and a final set of "output nodes". The nodes are connected amongst themselves by series of "paths", with some paths more "weighted" than others. Whenever a node is activated, it passes its activation value (some number, representing the relative strength of the activation) down the paths; each path, in turn, multiplies the activation value that passes through it by the path's "weight", so that the activation value on the other end of the path is some multiple of the original activation value. The receiving node then sums all of the activation values from its various incoming paths, and sends the resulting activation value further down the network, all the way to the output layer (see Figure 1 for a visual interpretation of the above description) [2].

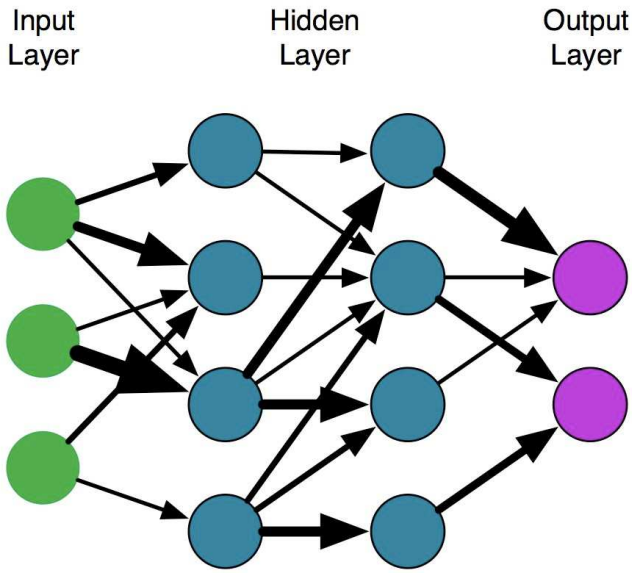

Figure 1: A very basic artificial neural network. The thickness of the arrows represents the weighted strengths of the connections.

From whence comes the power of the Neural Networks? From the interconnected nature of the nodes. For instance, unlike a traditional PC, a Neural Network does not need specific instructions or heuristics to help it identify a rose in a visual scene: just show it a picture of a rose (in the input layer), tweak the weights of the paths between the nodes until the output "rose" is activated, and then you're done! The next time that the same rose image is presented to the network, the output of "rose" will be obtained; more importantly, the next time that a *sufficiently-rose-like object is presented*, the network will still identify the flowers as a rose (especially if the alternatives *are sufficiently distinct* from a rose – for example, a bear or a fish). Moreover, the network can tolerate surprisingly high degrees of noise and occlusion – even if the whole lower half of the rose is obscured (the photographer didn't notice a hanging branch between himself and the rose), the network will still identify the image as a rose, because the non-obscured parts of the image are most rose-like compared to the other possible outputs[1] [2].

Though the Artificial Neural Network model is very straightforward and powerful, it is plagued by a very debilitating problem: namely, the difficulty of figuring out the necessary number of "hidden nodes" and the appropriate connection strengths between each and every node[2]. Teaching the network, therefore, requires very tedious and unnatural training, whereby weights are systematically adjusted until ALL of the appropriate inputs produce their corresponding appropriate outputs. Moreover, once the network is created and trained, it is

[1] For completeness sake, it should be noted that a given neural network would be trained on hundreds of images of the same object, and its internal "rose-like" representation would actually be a mix of all of the defining characteristics of being a Rose, in the Platonic Forms sense of the word.

[2] I had specifically glossed over this difficulty in the above paragraph, merely stating "tweak the weights of the paths between nodes until the output 'rose' is activate". The reality of the matter is that the "tweaking" is the *hardest* part of utilizing an Artificial Neural Network..

nearly impossible to teach it to respond to novel inputs, as doing so requires a near-complete re-training of a now-even-more complex system.

Thus, Artificial Neural Networks are NOT dynamic: they do not allow the system to adapt to novel input at a whim. ANNs are also fairly convoluted, as the different and unpredictable "weights" of the connections make it almost impossible to trace down an individual input node's contribution to the final output (indeed, there might not be any rhyme or reason to some of the connections, other than that they *happened* to cause the network to produce the appropriate response). With these criticisms in mind, it is therefore time to look at what makes DAN – the Dynamic Associative Network – different and more powerful that its predecessor ANN, the Artificial Neural Network described above.

A STEP AHEAD: THE DYNAMIC ASSOCIATIVE NETWORK MODEL

Dr. Anthony Beavers, Director of Cognitive Science at the University of Evansville, has spent the last several years working on a more dynamic and adaptable model of cognition than is currently embodied in Artificial Neural Networks [1]. Superficially, Dr. Beavers' Dynamic Associative Networks actually look rather similar to Artificial Neural Networks: both models are composed of a large set of nodes, with weighted pathways spanning between the nodes. Yet, DANs do not include the mysterious "hidden layer"[1], and, more importantly, the weighted connections within DANs are forged dynamically on the basis of the networks' experience, rather than on the basis of tedious systematic adjustments[1]. DANs also make no distinction about input and output nodes: each node has separate input and output channels, so *any* of the nodes could be activated, and likewise *any* of the nodes could be potential contenders for being the network's output.

At this point, it is necessary to introduce a bit of terminology. As stated above, beneath DAN's exterior lies a collection of "nodes", whereby a "node" is the model's representation of a given "item". The distinction between the two terms is that "items" – which might be words, numbers, pictures, etc. – exist in the input stream and are disjoint from each other, whereas "nodes" – one for each "item" – are more complex entities that carry properties and that cross-reference each other to form the basis of the model. Just as in the tradition neural networks, each node carries a certain "activation value", representing how actively it is contributing to the "current state" of the model, along with an "output value", representing how viable of a candidate the node is for being *the* output of the network.

When the model is initialized, it is a "Tabula rasa" (Latin) – a "blank slate". It contains no nodes, has experienced no long-term training, and is not in any "current state". As the model is "trained" by activating items in the input, it creates a new node for any non-existing item, and also creates a snapshot of the "current state" – that is, which nodes were active and to what degree – that gets implanted into each activated node. This implanted connection serves to solidify the transition from the previous state (that is, from the previously activated nodes) to the

[1] At least, not in the traditional ANN sense – more about this can be found in the "Results" section.

current state, ensuring that the next time that a similar input is encountered, the network will "remember" this past experience and steer its associations towards it. That is, the weighted connections between nodes in a DAN signify the number of times that certain nodes co-occurred together during training[1].

It should be noted that the model (it itself) does not specify the sphere of influence that a given item in the input has: Given a string of words, for example, there is nothing to say that the input should be broken down into X many chunks of words (where, within each chunk, each of the nodes would associate to each of the other nodes). The training input itself, however, is usually in the form of some chunk-like units (for text, one plausible unit is sentences; for images, the unit might be the actual image, encoded in pixels, plus whatever description – such as "rose" – that accompanies the image). Likewise, the model (in itself) does not specify to what degree items that are found in the input should be activated; for text, however, it might be logical to equally (fully) activate all of the words in a sentence, whereas for images it might be logical to activate each node to the extent that the given pixel was active in the picture (i.e.: if a node corresponding to the top-left pixel is fairly dark, it might get the value of 8 on a 10-point scale, whereas if it is only light-gray or white, the node might only get the value of 1 or 0).

A simple example of a DAN is shown in Figure 2. This particular network has been trained that *US_States*, *Alaska*, *Indiana*, and *Virginia* are all associated amongst themselves (during the "first" snapshot moment), and that *Indiana*, *Virginia*, and *Continental* are likewise associated amongst themselves (during the next snapshot moment). Given the above training, when *US States* is selected, *Alaska*, *Indiana*, and *Virginia* (along with the already-activated *US States*) all become the "next contenders" as the most likely associations (based on the very limited training data and an even more limited "current state"). When *Continental* is then selected in addition to *US_States*, only *Indiana* and *Virginia* remain as viable candidates for the most likely association. In "deducing" that only Indiana and Virginia correspond to *both* of the activated items, the network demonstrates some rudimentary cognitive abilities.

[1] This idea is very reminiscent of the well-known Hebbian neurological principle that nodes that "fire together, wire together" [4].

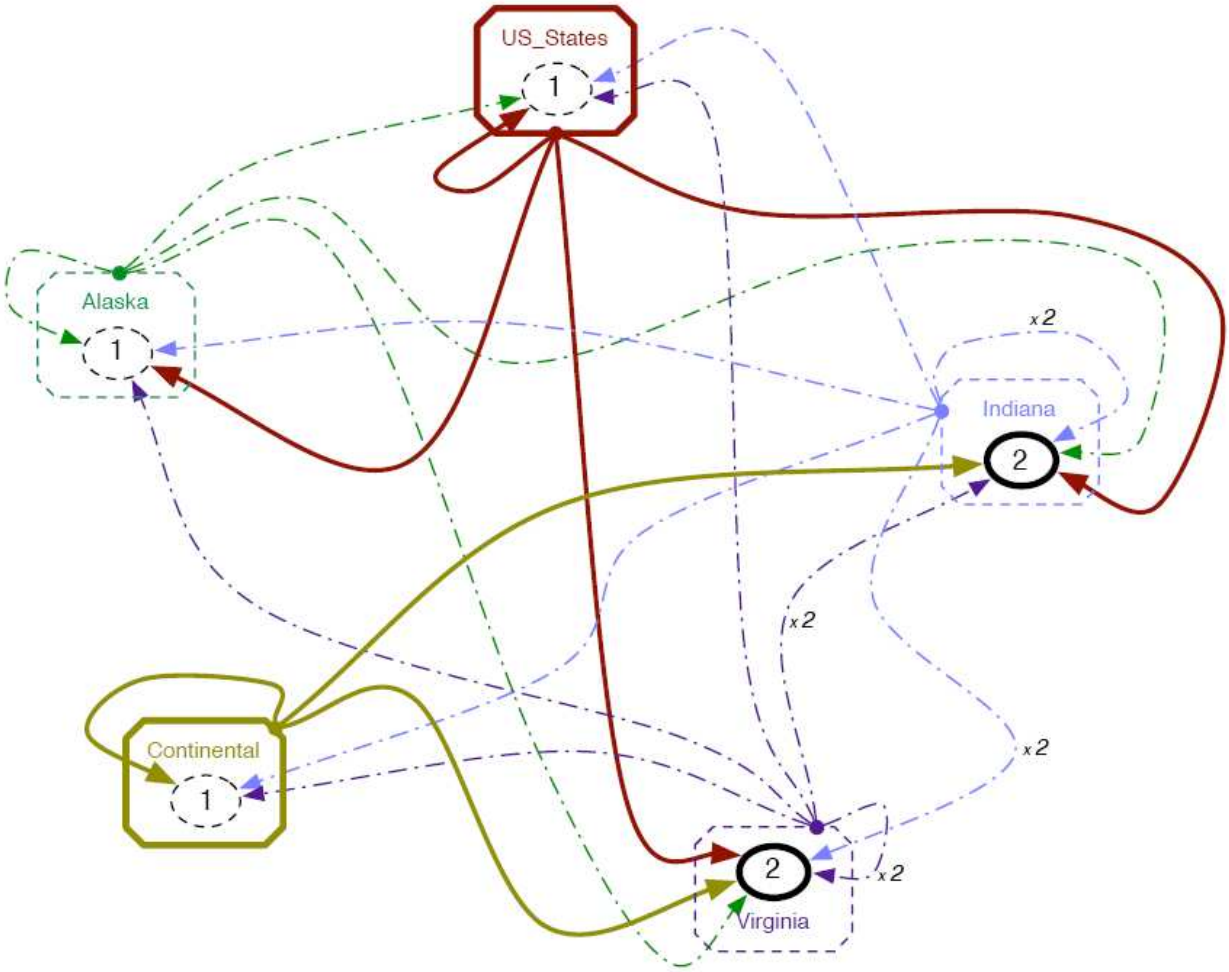

Figure 2: A simple DAN, trained on "US_States Alaska Virginia Indiana ||| Continental Indiana Virginia" [1]*, and currently being queried for the result of activating "Continental" and "US_States". Nodes are represented by rectangles, whereby the border of each rectangle is the "input" of the node, and the inner circle is the "output" of the node. The thick borders of Continental and US_States indicate that the two nodes are activated; moreover, the thick non-dashed lines emanating from the two nodes indicate that the US_States and Continental are contributing one unit to each of the nodes to which they are connected. The thin dashed arrows are pathways that were created during training, but that do not currently carry any activation (since their source node is inactive). With the exception of the pathways marked "x2" (which were created due to the dual occurrence of Virginia and Indiana in both training "sentences"), all of the pathways are of unit weight. The thick border around the outputs of Virginia and Indiana signify that both are nodes with the highest output, and therefore represent a "solution" to the network's query.*

[1] The triple-pipe sign (|||), due to its distinct and not-normally-occurring nature, was chosen as a sentence separator. Hence the network was actually trained on two separate (independent) sentences: first, "US_States Alaska Virginia Indiana", and later "Continental Indiana Virginia".

MODELING DYNAMIC ASSOCIATIVE NETWORKS IN SOFTWARE

My Senior Project task was to create a software suite for modeling and visualizing Dynamic Associative Networks[1]. The parallel nature of computations within DANs, along with the need to keep my model highly adaptable (as a project for Dr. Beavers' ongoing research, it was critical that my software suite could accommodate changes and/or new features for the DAN model) placed particular emphasis on careful software design.

DANs (and ANNs) are massively parallel structures: each node can be connected to a very large number of other nodes, and the change of activation on one node must result in a seemingly synchronous signal being sent down the network's numerous pathways. By the same token, the output of each node in a DAN is determined by signals being sent from countless *other* nodes, and it would be impractical for the network to re-calculate itself from scratch after every minor change in some node's activation. The computers that the simulation would run on, on the other hand, would be just regular PCs, which perform instructions in a sequential fashion, and would therefore have to *emulate* parallelism. Likewise, the program would need to implement some clever scheme to minimize re-calculations, as a full re-calculation as described above would run in $O(n^2)$, meaning that the refresh time would *quadruple* with each *doubling* of the number of nodes; at 10,000 nodes, a minor change in just one node would require as much as 100,000,000 re-computations!

My solution to both of the above problems culminated in designing a buffered change-propagating dependency-driven re-calculation scheme. Dependency-driven re-calculations ensured that only nodes that were directly impacted by a change were re-computed; if nodes A and B were mutually associated, but neither was associated with node C, then an increased activation on A would still have no impact on C, and hence there would be no sense in updating node C! Change propagation further simplified re-calculations, because instead of fully re-calculating the marked-to-be-recalculated nodes (which would involve finding all of the nodes that contribute to the given node, and summing their individual contributions), my change-propagating scheme simply passes the difference_from_former_value * connection_weight to each impacted node, exponentially reducing the complexity of the re-calculation (see Figure 3). Finally, buffering the changes – that is, waiting for all of the changes to cascade down the nodes before proceeding with other operations on the network – allowed the software suite to introduce all of the resulting changes in a seemingly singular "atomic" operation, which, for purposes of the DAN model, was identical to performing all of the changes in parallel. A similar buffering scheme was also used for re-displaying the network on the screen after introducing changes to multiple nodes: only when all of the changes had been performed would the computationally-intensive re-display and re-visualize routines get called.

[1] A website dedicated to the development of the DAN Software Suite – complete with detailed explanations of the underlying model, documentation, sample files, and a JAVA runtime, can be found at http://csprojects.evansville.edu/~mz13/.

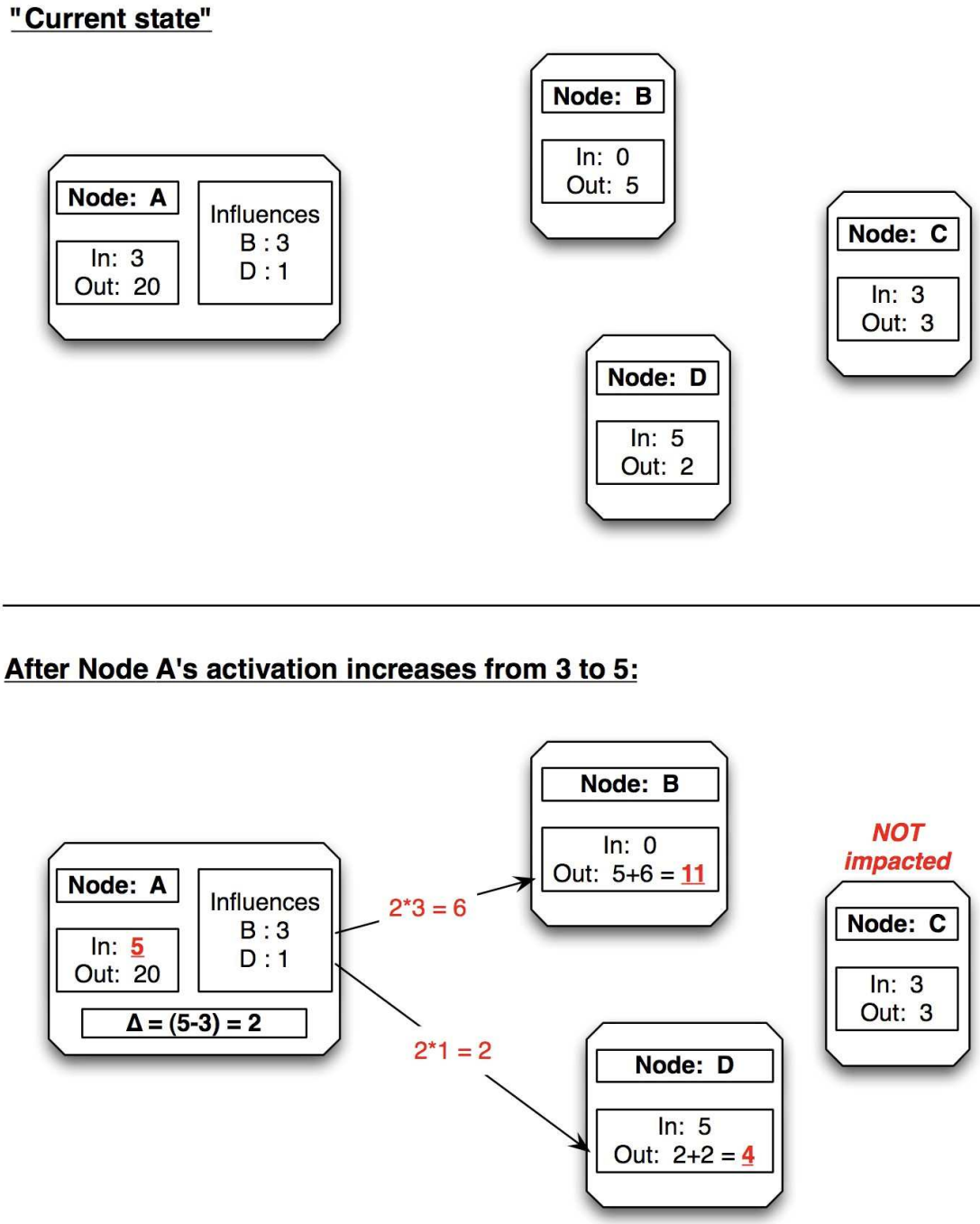

Figure 3: A change-propagating dependency-driven re-calculation scheme, whereby each node keeps a record of all of the OTHER nodes that get influenced by it (to reduce clutter, only Node A's list is show above). When a change in node A's activation (input) takes place, the difference_in_actiovation, multiplied by the weight of each connection, gets propagated to all of the dependent (influenced) nodes (B and D); nodes that are not dependent on A (namely, node C) do not get impacted, and are hence not re-calculated.

To facilitate the above scheme, at the very foundation of my program I created a Network class, which keeps a collection (technically, a <String, Node> hash table) of nodes. Each Node, in turn, keeps track of its input, its output, and a collection (technically, a <Node, ChangeableInteger> hash table) of other nodes that are influenced by the given node. Associations are initiated by the Network (for example, the network might tell node A to associate itself to node B and D), at which point node A adds nodes B and D to its collection, or, if the nodes are already present, increments the ChangeableInteger that represents the weight of the connection to the given nodes. When a node's activation changes, the node notes the difference between the previous value and the current value, and then informs all of the other nodes to adjust their outputs accordingly (namely, by the difference * weight). It can be proven by fairly straightforward inductive reasoning that as long as the original inputs and outputs of the nodes were valid, the above update algorithm would produce identical results to a brute-force re-calculation, except that the above algorithm would accomplish this task exponentially faster.

My software design also focused on separating the underlying DAN framework from the graphical user interface, and, indeed, on separating any components that did not necessitate being tied together[1]. The components (53 to date) are mostly centered around the Network (Nodes, Settings, and helper classes) and the User Interface Toolbar (which ties together all of the supporting windows, tabs, and saving and opening dialogues). Any "core" functions (including activation of specific nodes, creation of associations, and access to the nodes and settings) are routed through the Network, so that the user interface is essentially just a "wrapper" around a solid non-GUI foundation. The Network *does* maintain two-way contact with the user interface, however, for functions such flushing the network's contents onto the screen (i.e.: refreshing the display window or the visualization). The separation of components is critical for introducing new features into the software suite, or even for modifying the underlying DAN model (see the Results section).

RESULTS AND FUTURE WORK

The Dynamic Associative Network model is still a very young model, developed by Dr. Anthony Beavers over the course of only the last few years; the DAN software suite is younger still, having been developed by me over the course of only the last six months. Do we have any results to show that the DAN endeavor is worth pursuing?

The question is not as straightforward as it might appear, thought I do believe that the answer is positive. I will focus on three different aspects of our results: results of the workings of the software suite (i.e.: my senior project; the purely engineering aspects), results of the DAN model (i.e.: almost exclusively Dr. Beavers' work, though I did contribute one small but important aspect to the model, mentioned at the very end of this section), and the combined results in light of the overall goal to develop a "more robust and biologically-plausible cognitive model".

[1] Central to this notion is a Model-View-Controller design pattern from [3]. Other design patterns (most notably the Observer, Abstract Factory, and Proxy design patterns) from the same book were also used throughout my code.

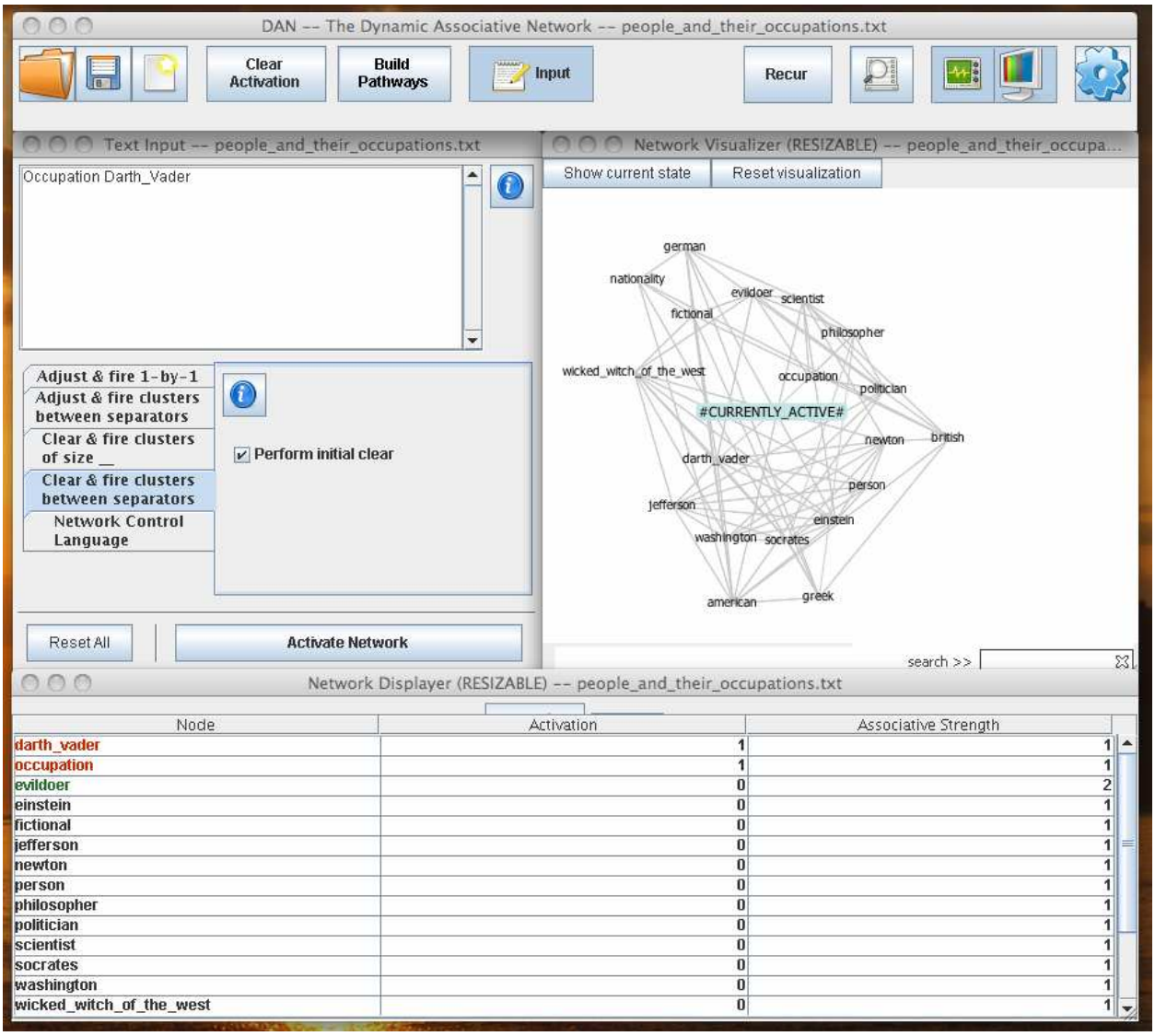

Figure 4: The DAN Software Suite, trained on simple associations between people (Einstein, Jefferson, Wicked Witch of the West, etc), their place of origin (German, American, Fictional, ...) and their occupations (Scientist, Politician, Philosopher, ...). The network correctly deduces that when "Darth Vader" and "Occupation" are selected (in red in the Network Displayer, and closest to the "#CURRENTLY_ACTIVE#" tag in the Visualizer), the node that best associates to BOTH items in the query is "Evildoer" (green in the Network Displayer; part of the tier-2 circle in the Visualizer). Each of the windows is independent of each other, governed solely by the underlying Network model that is at the very core of the software suite (and shown/hidden via the toggle buttons on the main toolbar).

The software suite – due to the engineering considerations that led me to find an efficient dependency-driven design the "core" framework, and to implement the program in terms of modular components – turned out to be very successful. From a performance benchmark point of view, the program can handle tens of thousands of nodes, and responds very quickly (i.e.:

learning a sample 4,569-word article, composed of 225 sentences, in under a second; this is a fairly remarkable achievement compared to a previous Excel-based DAN prototype, which would have taken over an hour to perform the same task). From a convenience point of view, the program's modular user interface allows the user to open only the windows that he/she wants to view (see Figure 4; in that figure, the open windows include the Toolbar, a Text Input window, a Network Visualizer, and a Display widow, but NOT the less-frequently-accessed Logger or Settings windows). Moreover, even the individual components inside the windows are sometimes separate classes, such as the different tabbed options in the Text Input window; that way, if a new type of "activator" is required, it can easily be added to the currently-available options. The least-finished part of the suite is the visualization component (still in early beta at the time of writing), but it will soon include color-coding and an auxiliary display to help the user visualize both the "grand scheme" of the network and the detailed individual relations of a selected node.[1]

The underlying DAN model, to the extent that Dr. Beavers has had a chance to experiment with it, has also shown some promising results. Powered by its dynamic associations, the model has demonstrated various rudimentary cognitive abilities (such as identifying members of particular categories or identifying similar objects; see Figure 4), the capacity for storing relational data (such as where an object is in relation to another object), and a potential for associating sequences of items (such as the digits in a phone number).

The DAN model has not gone unchanged, however. First of all, we have added the ability to create dissociations and one-way associations in addition to the default "full associations"; this allows for far greater flexibility in terms of teaching the network, permitting us to input causation-like associations (a causes b, but not vice-versa) and dissociations (the presence of a *inhibits* – that is, associates inversely on – b). Dr. Beavers also added a "Tuning" mode, whereby positive weights are capped at 1: this ensures that repeated associations of the same elements do not add carry undue significance. Tuning is particularly helpful for processing texts, where words such as "the" might frequently surround nouns, and yet should not count as anything more significant than other words in the sentence[2]. Finally, Dr. Beavers appended the concept of "Recurrence" to his original model, allowing the network to string its computation over time; through Recurrence, the Network takes its highest-output *non-activated* node(s) (shown in green in the Display window on the bottom of Figure 4) and activates them, while simultaneously decreasing the activation on the other active node (shown in red). Doing so allows the Network to "spread its tentacles" to grab a hold of an answer, or, in the case of some of the models, to store the current state in an internal buffer. In this latter case, Recurrence is somewhat analogous to an Artificial Neural Network's "hidden layer", with activation trickling from the input into this inner processing layer, before finally emerging in the output; but, unlike in the case of ANNs, these "hidden-layer" connections within a DAN are *still* forged on the basis of experience, rather than based on lengthy statistical adjustments.

---

[1] I should note that Visualization is the only NON-"home-grown" part of the software suite, employing an open-source visualization framework called PREFUSE. The framework is particularly helpful for aligning the nodes in a "Radial View" (though the complexity and the multi-purpose nature of the framework has also been the leading factor as to why the visualization is still in "beta").

[2] For example, given the learning "the boy woke up" and "the boy fell asleep", querying the word "boy" under Tuning mode would equally activate all the rest of the words, even though "the" and "boy" carry a double connection between each-other, due to their dual mutual occurrence during training.

It is not yet clear whether a Dynamic Associative Network is more robust than an Artificial Neural Network: there has simply not been enough time for Dr. Beavers and his colleagues to experiment with the model and to make use of the software suite that I have developed. The networks that Dr. Beavers *has* created thus far, however, were received with curiosity and some enthusiasm by a number of cognitive scientists and computer scientists at Indiana University at Bloomington, so the question of robustness definitely remains an open (and, in my mind, a hopeful) one [1].

As for biological plausibility, DANs certainly *do* outperform Artificial Neural Networks. Most critically, unlike ANNs, DANS are able to append to their existing learning "on the fly", much like everyday experience tells us about the cognition of animals and humans. DANs also follow the previously-mentioned Hebbian neurological principle that nodes that "fire together, wire together" [4]; this is in stark contrast to ANNs, which can wire nodes in an utterly unpredictable (and irrational) fashion. Initially, Dr. Beavers' DAN model did suffer from requiring an ever-growing list of dependencies between nodes (even when repeating previously-made associations), but I discovered that it was possible to amend this biological implausibility by utilizing *weighted* connections, which is how I've formulated the DAN model in my software suite and throughout this paper. Indeed, under all biological-plausibility criteria that I am aware of, Dynamic Associative Networks are simply more plausible than Artificial Neural Nets.

More research – indeed, *much* more research – will be necessary before the DAN model could even attempt to supplant the Artificial Neural Network model, if, indeed, that is ever to be the case. Yet I believe that the stage has been partially set, both through the refinement of Dr. Beavers' conception of Dynamic Associative Networks, and through my creation of a software suite that is capable of adequately modeling and visualizing DANs. As I prepare to graduate from the University of Evansville, I hope that my computer science successors, working under Dr. Beavers' continued supervision, will find the path cleared on their way to developing a more robust and biologically-plausible cognitive model.

SOURCES

[1] Beavers, Anthony F., Director of Cognitive Science. University of Evansville. Evansville, IN. Personal correspondence. Near-weekly meetings from April 2008 to March 2009.

[2] Churchland, P. M. (2000). *The Engine of Reason, the Seat of the Soul*. Cambridge, Massachusetts: The MIT Press.

[3] Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Indianapolis, Indiana: Addison-Wesley Professional.

[4] Hebb, D.O. (1949). *The Organization of Behavior*. New York: John Wiley.

BIOGRAPHICAL INFORMATION

Michael Zlatkovsky is a student at the University of Evansville, pursuing a dual-degree in Computer Science and Cognitive Science, with minors in Mathematics and Psychology. He has worked with Dr. Anthony Beavers on the Dynamic Associative Networks project for the past year, with additional guidance from his project mentor, Dr. Robert Morse. More information about the DAN Software Suite, along with a JAVA runtime and sample files, can be found on Michael's website at http://csprojects.evansville.edu/~mz13/. Michael will be graduating from U.E. this coming May, and will subsequently be enrolling in a Joint-PhD program (also in Computer Science and Cognitive Science), most likely at Indiana University.

