2015

Explorations Using Extensions and Modifications to the Oppenheim et al. Model for Cumulative Semantic Interference

Tyler Seip
Lehigh University

Explorations Using Extensions and Modifications to the Oppenheim *et al.* Model for Cumulative Semantic Interference

by

Tyler Seip

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Science

Lehigh University

May 2015

The thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

_____

Date

_____

Thesis Advisor

_____

Chairperson of Department

Acknowledgements

I would like to extend my deepest gratitude to both my advisor, Dr. Hector Munoz-Avila, and my co-advisor, Dr. Padraig O'Séaghdha. Their support and guidance were invaluable during this research.

I would also like to thank my family and friends for supporting me during my time at Lehigh, and always.

Table of Contents

List of Figures and Tables

Abstract

This thesis discusses extensions and modifications to a model of semantic interference originally introduced by Oppenheim *et al.* The first of the two networks presented extends the original toy model to be able to operate over realistic feature-norm datasets. The second of the two networks presented modifies the operation of this extended network in order to artificially activate non-shared features of competitor words during the selection process. Both networks were extensively tested over a wide range of possible simulation configurations. Metrics were developed to aid in predicting the behavior of these networks given the structure of the data used in the simulations. The networks were also tested for noise tolerance and duration of interference retention over time. The results of these experiments show resultant semantic interference behavior consistent with predictions over the parameter space tested, as well as high noise tolerance and the expected reductions in semantic interference effects as the networks were artificially aged. The new network models could be used as simulation platforms for experiments that wish to examine the emergence of semantic interference over complex or large datasets.

1: Introduction

It is well known that retrieval of a word from semantic memory affects future retrieval time for that same word. This is because the retrieval of a word also induces a learning event, which in turn changes the response time of subsequent retrievals. These effects have been classified into two cases, one positive and one negative. The first of these cases, referred to as *repetition priming*, improves both accuracy and response time of retrieval events for a target word the more it is accessed. The second of these cases, referred to as *cumulative semantic interference*, *reduces* the response time of retrieval events for words semantically related to an accessed word.

A computational model set out in Oppenheim, Dell, & Schwartz (2010) seeks to explain the underlying mechanisms causing these negative effects. They implement an artificial neural network that emulates picture naming experiments. By correctly modeling the semantic relationship between network inputs, they successfully produce network outputs that demonstrate cumulative semantic interference. In doing this, they claim that both repetition priming and semantic interference can both be explained as arising from an error-based learning process, and that ultimately it is error-based learning that is the driving force behind the changes in semantic memory retrieval time observed in experiments.

Their system works very simply. They simulate picture naming experiments by sequentially activating two inputs of the network corresponding to the “picture”, or word, that they wish to “show” to the network. They then apply a function to the network’s outputs to determine both the word that the network is outputting and an analog for the

response time of the network's output. This data allows them to determine if the network is producing cumulative semantic interference effects.

This implementation is theoretically useful – it shows that both repetition priming and semantic interference can ultimately be explained as the result of an underlying error-based learning mechanism. However, there are practical applications for a network such as this as well. A network like this could be used for simulating picture naming experiments if it were adapted to use more realistic inputs. There have been many feature norm datasets collected from human participants that could be used as inputs to a system such as this.

Because of its minimalist design, the network they implemented had a number of limitations. Word representation was limited to only two semantic features. Furthermore, words in this system can only share one feature between them. More realistic feature norms can have dozens of features, with complex semantic relationships. Additionally, learning in this network operates only on active inputs, which means that non-shared inputs of competitor words undergo no learning event, even though the word they correspond to is competing for selection.

This thesis seeks to both extend and modify the Oppenheim *et al.* architecture to support both: (1) generalized feature norm inputs, which allow for variable numbers of features, variable activation levels of these features, and arbitrary relationships between features of different words, and (2) the modification of connection weights for *all* inputs, shared or non-shared, belonging to competitor words, corresponding to semantically-oblivious learning events, while maintaining semantically dependent activation.

I first present background information necessary to understand the operation of neural networks and the basic principles of semantic interference in Chapters 2 and 3 respectively. I then describe the original Oppenheim *et al.* model in detail and present my extensions and modifications in Chapter 4. Empirical evaluations of the extended and modified models which seek to understand their respective behaviors over a wide parameter space are presented in Chapter 5. Finally, a summary of conclusions and suggestions for future work are briefly discussed in Chapter 6.

2: Neural Network Operation

2.1: Introduction

All of the models presented in this thesis are implemented as *artificial neural networks* (Haykin, 2004). Artificial neural networks are a well-studied and well understood statistical learning model whose architecture takes inspiration from biological neural networks. Sufficiently complex neural networks have been shown to be Turing complete, thus making them theoretically suitable for any computational task. All of the models in this thesis configure their underlying neural networks to act as a *classifier* (Duda & Hart, 2001).

A classifier, in general, takes a set of inputs, called *features*, and classifies this set (the *feature vector*) into one of many predefined *categories*. A neural network classifier achieves this via *propagating* the feature vector through its internal architecture and examining the resultant output. In all of the models presented, the *categories* correspond to *words* naming pictures in the picture naming experiments, and the *feature vector* is a set of feature norms describing this picture. The details of this procedure will be discussed in Chapter 4. Here, I present a short description of artificial neural networks in general.

2.2: Architecture

2.2.1: Overview

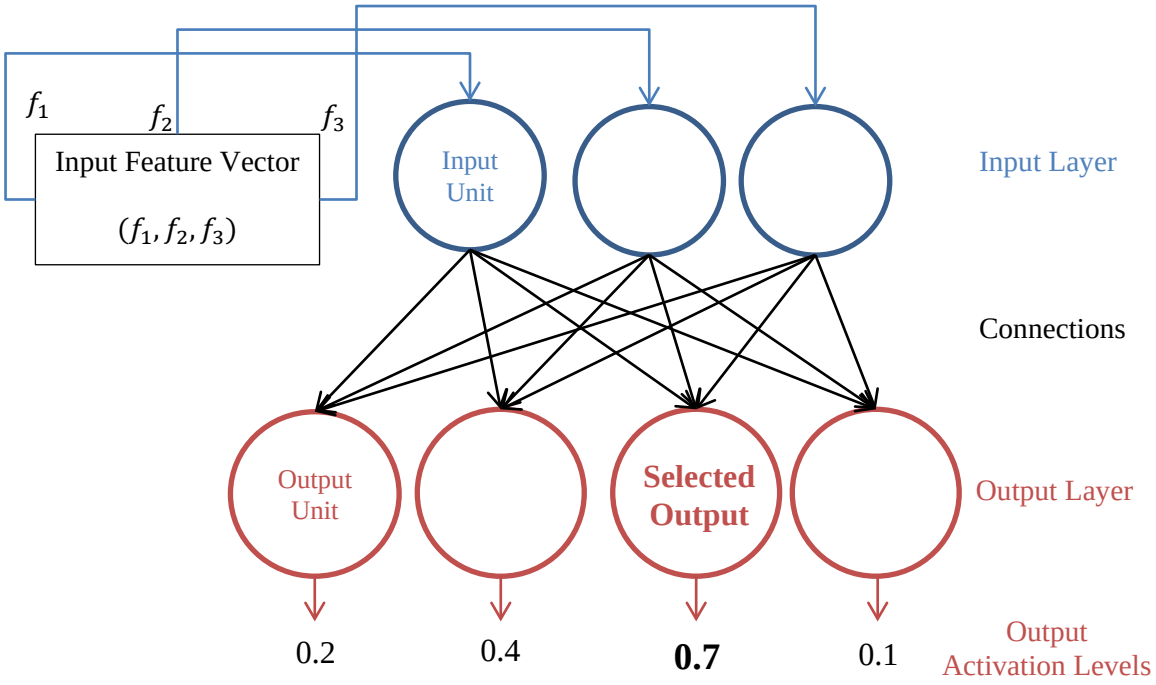

Illustration 1 - Basic Example of a Simple 2-Layer Network

An artificial neural network is fundamentally a directed graph. It consists of a set of nodes, or *units*, connected by a set of edges, generally referred to as *connections*. Loosely speaking, the units in an artificial neural network draw inspiration from neurons in a biological neural network; similarly, the connections draw inspiration from synapses. Generally, neural networks are organized into *layers*, and are often described by the number of layers they contain. For the purposes of this thesis, layers are composed of units which accept connections from the previous layer and originate connections to the next layer. Networks that do not follow this rule (i.e. networks that have connections running from a given layer to a *previous* layer) are referred to as recurrent networks. The

smallest nontrivial neural network, then, is composed of two layers. These layers will be referred to as the *input layer* and the *output layer* respectively, for reasons that will become clear shortly. If a neural network has more than 2 layers, the middle layers are collectively referred to as *hidden layers*. All of the networks presented, however, have only 2 layers, and so this chapter will focus on the properties of 2-layer, non-recurrent networks. Before we examine the architecture of neural networks as a whole, however, we must examine the operation of the network units and connections.

2.2.2: Unit and Connection Operation

As previously mentioned, units can have both *incoming connections*, from the previous layer, and *outgoing connections*, to the next layer. Units can be classified by the type of connections they have. *Input units* have only outgoing connections. *Output units* have only incoming connections. *Hidden units* have both incoming and outgoing connections. Thus, the first layer of a neural network, the *input layer*, is so named because it is composed solely of *input units*. Likewise, the *output layer* is composed only of *output units*. In general, every unit in a given layer is connected to every unit in its two adjacent layers: a set of incoming connections from each unit in the previous layer, and a set of outgoing connections to each unit in the next layer.

Each unit has an *activation level* which can be set in one of two ways. If the unit is an input unit, the activation level is directly set by the networks' input. If the unit is not an input unit, it calculates its activation level by applying a *network function,* $f(x)$, to all of its input connections. Most network functions commonly used take the *weighted sum* of all of the input connections and then apply a function to the result:

$$f(x) = K(\sum_i w_i g_i(x))$$

where w_i is the *weight* of incoming connection i, $g_i(x)$ is the activation level of the unit on the originating end of connection i, and $K()$ is a predefined function, referred to as the *activation function*, that generally maps the resultant output activation level to a value limited by the range of K

Common functions for calculating the activation level of a unit include the step function:

$$K(x) = \begin{cases} a \; if \; x < \epsilon \\ b \; if \; x \geq \epsilon \end{cases}$$

where ϵ acts as the *activation threshold*, and which has range (a, b)

the hyperbolic tangent function:

$$K(x) = \tanh(x)$$

which has range (-1, 1)

and the logistic function:

$$K(x) = \frac{1}{1 + e^{-x}}$$

which has range (0, 1)

Each of these functions have different desirable properties for constructing a neural network. For all neural networks discussed in this thesis, the logistic function is used as the activation function, in keeping with the Oppenheim *et al.* model.

The weighted sum of the input connections for the above expressions was calculated by multiplying the *weight* of each connection by the activation level of its source. In a neural network, every connection has a weight that determines the strength of "signal propagation" through it via this multiplication. Connection weights are thus

generally constrained to the range (0, 1). Connection weights can be changed – indeed, changing the weights of connections is the fundamental operation by which neural networks learn. I will discuss the mechanism by which these weights are changed in Sections 2.2.4 and 2.2.5.

2.2.3: Network Operation

As previously discussed, the overall network classifies a given input by doing the following:

1. Apply an input
2. Propagate the input through the network
3. Interpret the output

I will now explain each of these steps in greater detail.

Inputs to a neural network are feature vectors, which are composed of individual features. In general, a feature's value is simply a real value in the range of the activation function chosen. In order to apply an individual feature to a network, one simply sets the activation level of an input neuron to the feature's value. Therefore, to apply an entire feature vector, one must have an input layer with as many input neurons as dimensions in the feature vector. One then simply sets the activation level of each of the input neurons to the level of its corresponding feature. Once these activation levels are set, the input is allowed to propagate through the network.

Propagation is achieved through the hidden and output neurons' network functions. Once the activation levels of the input layer are set, the next layer (either hidden or output) allows each of its constituent units to calculate their own activation

level. This process is repeated layer by layer through the network until the output layer is reached. Since this thesis is concerned with only 2-layer networks, this process takes only one step: input layer to output layer.

Finally, the output is interpreted by examining the activation levels of the output neurons. The exact number of output neurons is determined by the application at hand. Oftentimes for classification tasks each output neuron will correspond to membership in a single class, so for example a binary classification problem would have two output neurons. The input feature vector is classified into the class represented by the output unit with the *highest activation value*. The networks in this thesis adopt this convention, but also use the values of the output layer units to calculate a separate function as well. This process, corresponding to decision difficulty in picture naming, is described in Chapter 4.

In order for this classification process to produce correct results, the weights w_i of the network's connections must be set correctly. Manually setting these weights in general would be nearly impossible. In fact, a neural network's internal structure is notorious for being difficult to understand even when correctly configured, let alone engineer. A learning algorithm is therefore adopted to configure the weights of these connections automatically.

2.2.4: Network Learning

In order for a neural network to automatically learn accurate and useful connection weights, it must be given a *training set* from which to learn. The training set is a set of training examples, which are pairs of the form (input feature vector, output class), where the given feature vector is defined as a member of the given class. The use

of a training set makes the learning algorithms discussed below *supervised learning algorithms*. A fourth step is then introduced into the operation of the neural network:

1. Apply an input
2. Propagate the input through the network
3. Interpret the output
4. Adjust connection weights

In step 4, the connection weights are adjusted via a *learning rule* – an equation that determines the change in weight for each connection. There are many different possible learning rules, and the choice of learning rule is often a question of engineering rather than mathematical analysis. Learning rules generally seek to minimize *network error* – that is, minimize the number of misclassified inputs. Networks can detect when they have produced an erroneous output during training by examining their own output and comparing it to the training class. Supervised learning rules will then adjust the network's connection weights in such a way as to move the network's output closer to the target training class. In this way, the network will be more accurate when the same input is presented again.

Before a network can be reliably used to classify inputs its weights must be adjusted in a *training phase*. The training phase presents each of examples from the training set in a random order once, allowing the network to adjust itself each time as governed by its learning rule. Additionally, it notes whether the network correctly classified the given output. It then repeats this process until a certain accuracy threshold is achieved, or for a fixed number of iterations. Each of these iterations through the entire

training set is called an *epoch* (the passing of one epoch corresponds to one iteration through the training set). The number of epochs required to train a network to a desired threshold is highly dependent on the structure of the network and the structure of the data. Sometimes, a given network architecture may fail to reach the desired accuracy threshold. We say that these networks *do not converge* for the given dataset. Networks that achieve the accuracy threshold are referred to as *convergent*.

2.2.5: The Learning Rule

The particular learning rule used by Oppenheim *et al.* in their network, and used in both of the networks presented, is the *Widrow-Hoff Rule*, tailored for the logistic activation function used in their constituent units. The actual implementation of this rule will be discussed in Chapter 4; however, a brief discussion of the theory supporting the rule is important, as we will see in Chapter 4 that one of the networks presented violates one of the assumptions of the rule.

The Widrow-Hoff Rule defines a cost function that measures how well the network has learned. It then seeks to minimize this cost function via the method of gradient descent (Widrow & Hoff, 1960). The cost function $E(w)$ is defined as follows:

$$E(w) = \frac{1}{2}\sum_{i}(d_i - a_i)^2$$

where w is the vector of all connection weights, d_i is the desired activation level of the ith output unit (supplied by the output of a training example), and a_i is the observed activation level of the ith output unit (calculated via propagation using the input of a training example)

Thus, the total "cost" or error calculated by this function is the sum of the square of the errors each of the output units is making. We modify each element of w, w_i by using gradient descent: calculate the gradient of $E(w_i)$, and subtract it from w_i. Once we do this for all connections, we will have changed the configuration of the network in such a way as to have *moved it towards a local minimum of* $E(w)$. This will minimize our error over time. We calculate the gradient of $E(w_i)$ with respect to w_i:

$$\frac{\partial E}{\partial w_i} = \sum_i (d_i - a_i) \frac{\partial(-a_i)}{\partial w_i} = -\sum_i (d_i - a_i)\, g_i(x)(1 - g_i(x))$$

and then use this to update the value of w_i:

$$w_i' = w_i - \eta \frac{\partial E}{\partial w_i} = w_i + \eta \sum_i (d_i - a_i) g_i(x)(1 - g_i(x))$$

In this expression, η is introduced as a scaling term which ranges between 0 and 1 called the *learning rate*. This term is introduced to control the adjustment that the network makes for each example. A small learning rate will cause the network to adjust more slowly, thus requiring more epochs. However, for complex datasets, small learning rates will often perform better than large learning rates, as the large jumps made by the network for disparate data can "overcompensate" and overshoot the minimum it was moving towards. This can lead to a cycle of overcompensation which converges at rate that Haykin describes as "excruciatingly slow".

One of the key assumptions made by this analysis is that:

$$\frac{\partial(a_i)}{\partial w_i} = g_i(x)(1 - g_i(x))$$

which is true for the logistic activation function. However, we will see in Section 4.3.4 that one of the networks actually *violates* this assumption – the gradient of a given output's activation level with respect to a given connection weight is dependent on *multiple* inputs. In order to compensate for this, I would need to re-derive the rule, introducing extra terms in the learning rule expression. See Section 4.3.4 for more details.

3: Semantic Interference

3.1: Introduction

All of the networks in this thesis seek to model *cumulative semantic interference*. In this chapter, I discuss some background information necessary for understanding what gives rise to semantic interference. I also discuss the blocked-cyclic naming paradigm, an experimental procedure used to measure interference effects which the networks emulate.

All of the experimental methods I will discuss are picture naming experiments, wherein participants are asked to name the subject of a picture. In general, a series of pictures of objects is presented to a participant, who is then asked to identify the shown objects. This is done in order to induce a series of *word retrieval* events, where the participant must retrieve the words that refer to the objects in question from memory. These are often referred to as *word production tasks*.

The central focus of these studies is to gain insight into the structure of memory, including memory of meaning-to-word mappings. With clever experimental design, it is possible to begin to understand how related memories are stored and how those memories change over time by examining the way in which word retrieval events occur.

However, one cannot expect to simply watch neural activation in response to these pictures and expect to gain insight into the word retrieval process. Instead, a number of more easily understood metrics are examined. One metric that is commonly used is the *word retrieval time*, which is the amount of time a word retrieval event requires to complete. The experiments which seek to measure word retrieval times will measure the *response time* of the participant – the time the participant takes to

successfully identify the subject of the picture by its name. They will then use the participant's response time as a proxy for word retrieval time.

We will see that response time analysis can yield some interesting insights into the structure and behavior of memory.

3.2: Insights from Response Time

The first major effect that can be observed from measuring participants' response time to pictures is *repetition priming*. Suppose one measures the response time of a particular individual's first exposure to a picture ***p***. If ***p*** is then presented again to the participant, we will tend to see a *reduction* in response time to ***p***. Furthermore, this reduction in response time can last on the order of days or weeks; the participant will answer more quickly for successive presentations of ***p***, even when long interstitial periods between the naming sessions are instituted (Brown, 1979). This is the core of repetition priming: that *repeated exposure to a stimulus will improve response time and accuracy*. Furthermore, these improvements last a long time.

Clearly, the participant's word retrieval process must be changing over time to accommodate the observed changes in response times. Lachman & Lachman (1980) note that these changes clearly cannot be caused by a transient effect – to attribute them thusly would ignore their long lasting effects. It must then be concluded that some sort of long term modification occurs in response to this stimulus processing. In other words, it must be concluded that this word retrieval event is also a *learning event*. The notion that all word retrieval events are also learning events plays a central role in both the original Oppenheim *et al.* model and the modifications I present.

The second major effect one can observe from measuring response time is *semantic interference*, the central concern of this thesis. Semantic interference is complex and multi-layered, and arises from a number of competing processes. The particular effects that I am concerned with, and that were modeled by Oppenheim *et al.*, however, are as follows: *for a given target picture **p**, repetition priming of its semantic competitors results in slower word production for **p***. Unpacking this a bit, suppose one has a set of pictures that are all members of the same *semantic category*, e.g. animals. These pictures are presented in sequence. Over time, response times will tend to *increase* for each picture presented as compared to a baseline response time. This baseline response time is generally measured by placing the picture in a context wherein it is preceded by pictures that are *not* members of the same semantic category, and measuring the response time in this condition.

This net increase in response time is generally attributed to increases in competitor availability caused by the repetition priming of those competitors. Essentially, the retrieval is slowed down not by an *absolute* decrease in availability of the target word, but rather a *relative* decrease in availability of the target word compared to its strengthened competitors (Wheeldon & Monsell, 1994).

This type of semantic interference is known as *cumulative semantic interference*, as its effects build up over time, and last for an appreciable period – on the order of minutes in experiments and potentially much longer. This is due to its dependence on repetition priming, whose effects are known to last for a very long time as well. Other

types of semantic interference (*noncumulative semantic interference*) will not be discussed in this thesis.

3.3: The Three Principles of Howard *et al.*

If we wish to model semantic interference, we clearly must have a mechanism that simulates repetition priming. Furthermore, we must simulate homogeneous and heterogeneous conditions as described in Section 3.2. Finally, the notion of "semantic competitors" must be modeled. These requirements are corroborated by Howard *et al.* (2006), who give a set of three necessary principles that must be implemented in any system that seeks to model semantic interference: *shared activation*, *competitive selection*, and *repetition priming*. In Section 4.1.3, I will discuss precisely how the Oppenheim *et al.* model and my extensions fulfill these three principles. For now, I will describe the first two principles in greater detail, as I have already described repetition priming in Section 3.2.

Shared activation in this context refers to the particular way in which homogeneous and heterogeneous conditions are implemented. It must be the case that when a picture is presented to the model, two things must occur. First, the model must select the correct word that identifies the picture; second, the model must also consider words that are semantically related to the correct word. In other words, the presentation of a picture must activate all words that are semantically related to that picture to some degree. The structure of a neural network allows for easy implementation of this

requirement if words are described as sets of related features which can be shared among other, semantically related words; more details on this can be found in Section 4.1.3.

Competitive selection is tied into the previous requirement. With the shared activation principle implemented, we know we get a set of partially activated candidate words upon a picture's presentation. The competitive selection criterion stipulates that these candidate words must *delay the production of the correct word*. In other words, if the target word and the other competitors have very similar activation levels, this must result in slower production of the target word overall. This criterion is implemented in the networks presented by using a *boosting mechanism*, further described in Section 4.1.2.

3.4: The Blocked-Cyclic Naming Paradigm

Many experimental setups have been designed in order to produce semantic interference in a controllable manner. There are two main paradigms generally used, however: the continuous naming paradigm and the blocked-cyclic naming paradigm.

The continuous naming paradigm (originally described by Brown, 1979) presents a non-repeating stream of semantically related pictures. It also often incorporates non-semantically related pictures throughout the stream, to counteract a number of short-term priming effects that otherwise interfere with the semantic interference effects being examined. This paradigm was explored by Oppenheim *et al.*, but will not be simulated in this thesis. However, the extensions I describe are capable of running experiments of this style.

Instead, I focus on the blocked-cyclic naming paradigm. In the blocked-cyclic paradigm, a small repeating set of pictures is presented to the participant in random order.

The participant identifies each as quickly as possible. One presentation of the entire set (in random order) is called a *cycle*. The presentations are repeated for a set number of cycles. Once the set number of cycles is complete, the experiment can be repeated again for a variable number of *blocks*. Before any of this occurs, the participants are allowed to familiarize themselves with all of the pictures (Frazer *et al.*, 2014).

The design of the sets of pictures in these experiments is crucial. First, one constructs a number of homogeneous sets of pictures, e.g. a set of birds, or a set of vegetables. Then, an equal number of heterogeneous sets are constructed, by selecting one element of each homogeneous set and collating them together. This ensures that the heterogeneous sets are both uniform and equally related to the homogeneous sets. In this way, the amount of possible unintended semantic overlap between the homogeneous and heterogeneous conditions can be minimized.

The networks presented in this thesis were tested using simulations of the blocked-cyclic paradigm. The training phase of the network corresponds to basic vocabulary acquisition. Real participants would already know the language they were expected to identify the pictures in. The testing phases are then executed on each of the different conditions, constructed exactly as described above. For this step I use separate clones of the network for most simulations.

4: Extensions to the Oppenheim *et al.* Model

4.1: A Short Description of the Original Model

4.1.1: Motivations

As previously discussed, the computational model set out in Oppenheim, Dell, & Schwartz (2010) was designed to show that both *cumulative semantic interference* and *repetition priming* result from a unified underlying error-based learning process – that they are, so to speak, "two sides of the same coin". The authors note three necessary principles for cumulative semantic interference, originally outlined by Howard *et al.* (2006): *shared activation*, *competitive selection*, and *priming*. Any system modeling a semantic interference-like effect must include mechanisms that effectively implement each of these three principles. With these principles in mind, the authors implemented a two-layer neural network with strictly feedforward connections, whose neurons have logistic activation functions. This network was designed to simulate experiments from the blocked-cyclic naming paradigm. I will first describe the specifics of their network's implementation (hereafter referred to as the *baseline network* for convenience), and then justify the implementation as adequately fulfilling all of the above outlined requirements for the emergence of semantic interference.

4.1.2: Implementation Details

The output units of the neural network map to *words*, i.e., fundamental elements of lexical memory retrieval with implicit semantic content. In general, words can be thought of as picture names in the blocked-cyclic naming paradigm. Input units of the

network represent *features*, i.e., semantic descriptors of the set of words. These units loosely correspond to adjectives or descriptors one might use to describe the subject of a picture. A word is uniquely described by a set of features, and is thus decomposable into its constituent features. For example, the word *whale* might be described by the feature set {mammalian, aquatic}. In the Oppenheim *et al.* implementation, each word is limited to only two features – this means, in general, the maximum number of words that can be described by a feature set of size n is given by:

$$words_{max} = \binom{n}{2}$$

This count assumes that no two words can share both features – if this were the case, the words would be identically defined and would be indistinguishable.

It is important to note here that the only assumption built into the model via the feature and word representation is that, at some level, words are *de facto* represented as combinations of decomposable units that are reused across other words. The loose correspondence between "adjectives" and features that is used here for convenience is therefore not necessary for this model to be valid – the "adjectives" could just as easily be sub-concepts, or qualia, or bundles of co-activated neurons – the only important thing is that the units are reused across whatever corresponds to words in the system in question.

Each input unit is connected to each output unit by a connection with initial weight 0. Each *connection weight* is updated at each time step by a specially tailored variant of the Widrow-Hoff learning rule:

$$\Delta w_{ij} = \eta\big(a_i(1 - a_i)(d_i - a_i)\big)a_j$$

where Δw_{ij} is the change in strength of the connection from node j to node i, a_i is the activation level of node i, d_i is the target activation level of node i, and η is a configurable learning rate parameter that governs the step size of the gradient descent algorithm used to minimize the network error.

The only difference between this rule, used by Oppenheim *et al.*, and the unmodified Widrow-Hoff (or delta) rule is the addition of the $a_i(1 - a_i)$ term. This term simply weights changes to output nodes at the brink of indecision (i.e. whose output is approximately .5) more heavily than changes to output nodes whose outputs are very close to either 0 or 1 (i.e., fully activated or fully inactivated). It is a direct result of the use of the logistic activation function – this was derived in Section 2.2.4. Because of the a_j term, which will be more fully discussed later, the weights of connections emanating from inactive input nodes are invariant.

The actual activation levels of each output node are calculated using the logistic activation function given in Chapter 2. The activation levels of the input nodes are of course manually set to reflect the features of the virtual "picture" to be named by the network. For example, to "show" a picture of the word "dog" to the network, where the word dog is described by ({mammalian, furry}, dog), one would set the activation levels of the two nodes representing the features "mammalian" and "furry" to 1, and leave all other nodes' activation levels at 0.

Because this network was designed to simulate blocked-cyclic naming paradigm experiments, its operation is unusual in that we look to evaluate its performance *over time* in order to reflect the subject's performance over multiple blocks in the cycle.

Additionally, the parameter we are most interested in measuring is not the actual output of the network, but rather the relative strength of that output against the other possible outputs. This relative strength acts as a proxy for naming time, which itself is used as a proxy for word retrieval speed. Because of this, Oppenheim *et al.* define the "time", $t_{selection}$, taken by the network to distinguish the strongest output to be:

$$t_{selection} = \log_{\beta}\left(\frac{\tau}{a_i - a_{others}}\right)$$

where β is a free parameter called the *boosting rate*, τ is a threshold value to boost to (which the authors set to be 1), and a_{others} is the average activation of all of the outputs not selected

This equation, which is computationally equivalent to multiplicatively boosting the activation level of each outputted word until the threshold τ is reached, outputs the number of *boosts* (the value of $t_{selection}$) required to reach this threshold – this number is used in place of response time for the experimental simulations. Modifying the boosting rate logarithmically scales the calculated values of $t_{selection}$. For my purposes, the value of the boosting rate must be greater than 1 – I use a boosting rate of 1.06.

As with all neural networks, a training phase must be executed before any simulations are ran in order to initialize the connection weights such that correct responses result from a given input. Because I am concerned with measuring a phenomenon that occurs over time, it is extremely important that all comparisons between networks (i.e. across experiments) occur between similarly trained networks. Oppenheim *et al.* solve this problem by training each network for a constant 100 epochs.

Later, we will see that for extensions to this model, modifications must be made to this training period. However, because the networks used in the original experiments are all the same size, the use of a constant training period is a reasonable simplification for the original network.

Included in this network and all of the extensions of the network yet to be presented is a noise parameter, θ. In this case, θ selects the standard deviation of a normal distribution (with mean 0: $v_{noise} \sim N(0, \theta)$) from which noise vectors are sampled. Thus, this parameter serves to control the magnitude of random perturbations affecting the weights propagated throughout the system. For low values of the noise parameter, 100% network accuracy can be achieved. For higher values, the network's ultimate accuracy asymptotically approaches a maximal value. A network's robustness in the face of noise is an important parameter to explore. All real-world examples of systems that produce semantic interference (e.g. the human brain) are also generally very noisy. This will be examined later in Chapter 5.

4.1.3: Fulfilling the Three Principles of Howard *et al.*

The network as outlined above implements *shared activation* via its feature-based representation of target output words. As mentioned previously, because words are abstracted as sets of features, and because those features can be shared across words, activation of an individual feature tends to activate more than one word simultaneously – in this way, the network implements a shared activation mechanism.

Competitive selection is achieved by the network via the boosting mechanism. If a particular target word is activated, the outputted boost count is calculated by taking into

account the average activation of all competitor words, thus ensuring that increased activation in competitors leads to an increase in measured "response time" – precisely the definition of competitive selection. It should also be noted that an increase in activation of competitors here necessarily corresponds to an increase in the activation of extraneous features that do not belong to the target output. In this case, inhibitory connections from the extraneous features to the target output will reduce its overall activation, which will relatively increase the activation of its competitors, further realizing the competitive selection mechanism.

Finally, *priming* is achieved via the implementation of the learning rule. Successful access to a target word ***o*** will necessarily cause the learning rule to update the connection weights of the network such that the word in question will be more strongly activated in future epochs by directly strengthening the connections from the inputs. Furthermore, access to other words will *weaken* the connections from these words' inputs to ***o***, which over time will have the net effect of decreasing the net activation of competitors for ***o***, facilitating access to ***o***.

In implementing all three of the Howard *et al.* principles, the Oppenheim *et al.* framework demonstrates a capacity to exhibit cumulative semantic interference.

4.2: Direct Generalization

4.2.1: Motivations

The original Oppenheim *et al.* model imposes two important constraints on the possible inputs to their network: first, all input values are binary – an input unit is either

fully excited (1) or completely dormant (0); second, each output has precisely two input features that specify it. Thus, an input-output pair for the original network is fully described by a non-weighted list of two unique features and one target output word, e.g. ({mammalian, four-legged}, dog). This approach, while effective, is highly restrictive. A more general model would have words with more than 2 features, and would allow these features to be variably activated – not simply on or off. Indeed, there have been many attempts to collect realistic feature-norm data for objects from humans – none of them describe a real object as a simple non-weighted set of two features. In McRae *et al.* (2005) we find a rich feature production norm data set meant for experiments of precisely this style. With datasets like this in mind, I seek to generalize the original Oppenheim model for use in modeling semantic interference over a more general parameter space, where I can model both (1) the number of features per word, and (2) the activation levels of each of the input features.

4.2.2: Implementation Details

Consider the case of specifying a word such as “penguin”: while it is indeed taxonomically a bird, it is likely less central to one’s conception of “bird” than, for example, an eagle. Feature production norm datasets such as the one provided by McRae *et al.* capture these relationships by assigning each feature a value derived from their respective production frequencies. Thus, each word (*concept*) in the McRae *et al.* dataset is described by a set of *ordered pairs* of the form (feature, value). These values range from 1 to 30 and reflect the number of participants who listed that particular feature for that particular concept. An example norm for the concept “ball” is reproduced below:

Feature	Value
used_by_bouncing	19
is_round	17
used_for_sports	13
used_by_throwing	8
used_for_playing	8
different_colours	6
is_fun	6
is_hard	5

Table 1 - Feature norms for concept "ball"

Norms of this form suggest an obvious way to generalize the original network for my purposes: simply include an input for each feature as before, and then activate each input feature with strength proportional to the corresponding norm weight. This suggests the following mapping from McRae *et al.* feature norms to input activation levels:

$$a_j = \frac{v_j}{\sum v}$$

where a_j is the input unit corresponding to a particular feature whose value is v_j and $\sum v$ is the sum of the values of all of the feature norms for that word

However, this procedure only works if the number of features per word is fixed. In order to further generalize this procedure, we must normalize the length of the resultant vector of input activation levels. This ensures that extra weight is not afforded to vectors of higher dimension (i.e. words with more features) – and it also allows us to use Euclidean distance as a measure of dissimilarity between two words, as all words in this system are represented as unit vectors rotated about the origin of a high-dimensional feature space.

The final normalization routine used to map the McRae *et al.* norms to the input units is given by the following pseudo-code:

```
procedure normalize(input norms, output levels):
//Find the minimum norm value
min = norms[0]
for i = 0 to norms.length:
    if(norm[i] < min):
            min = norm[i]
end for
//Find the sum of the squares of the weights,
//normalized by the minimum value
sum = 0
for i = 0 to norms.length:
    sum = sum + (norm[i]/min)^2
end for
//Find the inverse sqrt of this sum
sum = 1.0/sqrt(sum)
//Use this and the minimum to normalize each weight
for i = 0 to norms.length:
    levels[i] = (norms[i]/min)*sum
end for
return levels
```

Figure 1 - Normalization Pseudocode

This procedure operates very simply. We first find the minimum value among the norms. We scale each norm by this value, and then find the overall length of the resultant

vector. Finally, we normalize the vector by multiplying by the inverse of the calculated norm. This ensures that the resultant vector is a unit vector, and that each component maintains their relative strength from the original norm set

With this normalization routine, along with the McRae norms, I hope to show that evidence of semantic interference can be found in simulations that reflect a more realistic experimental structure and dataset Furthermore, I wish to explore the performance and behavior of the network when I systematically vary the internal structure and overall size of the dataset Discussion of these results can be found in chapter 5.

4.3: Modifications to the basic Oppenheim *et al.* architecture

4.3.1: Motivations

The learning rule of the original Oppenheim network, in keeping with the normal rule for gradient descent error minimization, scales the weight change of each connection per update by the activation level of the input unit it emanates from. While this leads to a network that is easy to understand and analyze, it also lends the following property to the system: if an input unit is not being excited, no changes can occur to the weights of any of its connections. This means, effectively, that the input features of competitor words are never “in play”, so to speak, unless those inputs happen to be shared across words (and thus currently active).

I felt this was unrealistic behavior. When the network is selecting a word to output, it evaluates each candidate word against a set of competitor words. I reasoned that evaluating the strength of each of these competitors constituted a retrieval event. In keeping with the notion that “retrieval events are also learning events”, each of these

outputs, whether they are ultimately selected or not, should be treated equivalently. Therefore, *all* of the input features of both competitor words and the selected word should be "in play", shared or non-shared (O'Séaghdha *et al.*, 2013).

I also reasoned that when presented with a set of words in close proximity, like in the blocked-cyclic naming paradigm, a human participant would consider not only features of the particular word being shown, but also remnants of the features of other homogeneous words presented, and features that themselves were semantically related to the features belonging to the word shown.

Because of this, I sought to modify the network to accommodate changes in connection weights for inactive inputs with the following constraints in mind: (1) the learning rule should remain unchanged; (2) the basic network architecture (2 layer) should remain unchanged, and (3) any resultant modified network should show cumulative semantic interference across all datasets over which the unmodified network can.

In keeping with these principles, I wish to excite additional input units such that their connections are modified as well. These input units should in some way be related to the baseline input vector – I do not wish to arbitrarily excite input neurons. Arbitrary excitement would either be indistinguishable from noise, or indistinguishable from a *different* input – neither of which are useful modifications to model. There are two ways of exciting secondary input units without significantly altering the network architecture. I dubbed these two variants *temporal* and *spatial* excitement paradigms.

The *temporal* excitement paradigm seeks to excite secondary inputs as a function of their previous states. This would, in effect, model *temporary priming*, and is in fact mentioned by Oppenheim *et al.* in their paper as a relatively *weak* explanation for cumulative semantic interference. One possible way to model this would be to introduce a *residual activation* parameter, α, which ranges between 0 and 1. The inputs of the system at time step *t* would then be given by:

$$a_{j_t} = \max(\alpha * a_{j_{t-1}}, \delta_j)$$

where δ_j is the applied input at time step *t*

Clearly, $\alpha = 0$ gives us no residual activation and thus results in no change. Some cursory tests were performed using this paradigm, and for almost all values ($\alpha \geq 0.01$) I found highly erratic and incorrect outputs from even simple simulations. This does not mean that such an effect is therefore unrealistic – only that implementations of it that simply decay each input by a constant factor at each time step fail to produce useful results. Because this avenue did not seem particularly fruitful, I examined the *spatial* excitement paradigm.

The *spatial* excitement paradigm seeks to excite secondary inputs as a function of *other* currently excited inputs. Because inputs in this paradigm can influence the activation of other inputs, a *spatial ordering* of the inputs can be observed for a given input (e.g. unit i activates unit j activates unit k…), hence the "spatial" moniker. The most general system in this paradigm instantiates extra connections from each input to every other input as well as the connections already seen. Presumably, features themselves can sympathetically activate or inhibit one another if they are semantically

related (e.g. winged might activate aerial). Indeed, if I wish to activate the non-shared features of competitor outputs, a procedure such as this becomes necessary. A *competitor* output is distinguished from the *selected* output by virtue of its activation level. If its activation level is not the maximal level across all outputs, then it is a competitor. Because a competitor output's activation level is entirely determined by the input activation levels, I must activate the desired non-shared features as a function of the activated features.

This raises the question: how do I assign realistic weights to these inter-input connections? Generally, the connection weights in a neural network are reached via the learning process. However, the assumptions built into the Widrow-Hoff learning rule (as implemented by Oppenheim *et al.*) do not hold in network architectures more complex than the 2 layer feedforward network they implemented. Clearly, unless I change the learning rule, these connection weights cannot be accurately or meaningfully learned in the same way the normal input-to-output connections are.

I have no data on the semantic relationships between features. However, I do know which output words are semantically related – and I know which features map to these words. This suggests a method for implementing the above changes without seriously modifying the underlying architecture or operation of the network. This method will be discussed in the next section.

Ultimately, it is this second modification that I decided to more fully explore. In chapter 5, each simulation, when applicable, will be presented as run on the unmodified,

generalized Oppenheim-style network from section 4.2 *and* as run on the modified network presented in this section using the *spatial* excitement paradigm for comparison.

4.3.2: Implementation Details

Suppose I excite a particular set of inputs, ignoring the effects of noise for a moment. This will select an output, ***o***. The output ***o*** will have a number of competitors, some strongly activated, some weakly activated. All of these competitors will share at least one feature with ***o*** – I know this, because otherwise they will not be activated at all. When the network updates its connection weights, all of the connections from all of the features of ***o*** will be updated. However, only the *shared* features will be updated for all of the competitors of ***o*** – even though they are (weakly) activated as well! In other words, the network essentially distinguishes between the single output excited "in actuality" and outputs excited "sympathetically" when modifying its internal state. This is incongruous with the notion that retrieval events are also learning events – there is retrieval without learning occurring here. I therefore seek to apply learning to the *all* features of activated outputs, not just the selected one.

Additionally, suppose a subset of the set of input features for a particular word is excited, e.g. excite {mammalian, four-legged} for the input-output pair ({mammalian, four-legged, furred}, dog). One can safely assume that a properly trained network will reasonably excite the "dog" output unit given this "partial" input (assuming no other input is closer).

If the network recognizes that it is currently viewing a dog from solely the partial input, perhaps we can *infer* the additional, unseen features from the presence of the

features that are activated. In other words, perhaps we can form predictive rules of the form ({mammalian, four-legged} → {furred}) by examining the *output* of the partial input. This would allow us to artificially excite input units that *should* be in play yet are not, given the structure of the input the network expects. In this way, we can reasonably and programmatically excite secondary features that are semantically related to the primary activations.

This also allows us to update the weights of both the shared and unshared features of competitor output nodes – by activating the shared node, this method will automatically excite the unshared nodes belonging to the competitors as well.

The method used for producing these secondary activations is given below in pseudo-code. It takes in a set of primary activations and outputs a new set of activations that include the primary activations as well as any secondary activations calculated using the method:

```
procedure exciteSecondary(input primaryInputs, output
newInput):
//first, calculate the natural output of the given //inputs
propagateInputs(primaryInputs)
foreach output in outputLayer:
     for i = 0 to primaryInputs.length:
          newInput[i] += output.level *
               connectionWeight(inputLayer[i],output)
     end for
end for
for i = 0 to newInput.length:
     newInput[i] = 1/(1+e^-newInput[i])
     newInput[i] = max(newInput[i], primaryInputs[i])
end for
return newInput
```

Figure 2 - Secondary Activation Pseudocode

First, we activate the outputs as normal. Then, we temporarily *reverse* the directions of each connection (i.e. features-to-words connections become words-to-features). We treat the output activations, calculated in step one, as *inputs*, and propagate the activations back to the feature layer. We then take the maximum of this new calculated input set and the old primary inputs, and use this as our new input.

4.3.3: Implementation Analysis

We can show that this procedure is roughly equivalent to instantiating additional connections between features:

From the first step, we know the value of each output node is:

$$o_i = \frac{1}{1 + e^{-\sum_j w_{ij} a_j}}$$

where w_{ij} is the weight of the connection from input j to output i, and a_j is the value of input j.

After the connection reversal step, we have the value of each input node as:

$$a_j = \frac{1}{1 + e^{-\sum_i w_{ij} o_i}}$$

If we use the Taylor expansion of the output node's value as an approximation, we get:

$$o_i = \frac{1}{2} + \frac{\sum_j w_{ij} a_j}{4} + O\left(\sum_j {w_{ij} a_j}^3\right) \approx \frac{1}{2} + \frac{\sum_j w_{ij} a_j}{4}$$

We can safely neglect the $O(\sum {w_{ij} a_j}^3)$ term, as $0 < w_{ij} a_j < 1$.

Substituting this into our expression for the input excitation, we get:

$$a_k = \frac{1}{1 + e^{-\sum_i w_{ik} \left(\frac{1}{2} + \frac{\sum_j w_{ij} a_j}{4}\right)}} = \frac{1}{1 + e^{-\sum_i \frac{w_{ik}}{2} + \frac{w_{ik}}{4} \sum_j w_{ij} a_j}}$$

Then we set:

$$a_k = \max(p_i, \frac{1}{1 + e^{-\sum_i \frac{w_{ik}}{2} + \frac{w_{ik}}{4} \sum_j w_{ij} a_j}})$$

where p_i is the applied input at i.

Suppose we had connections from each feature to every other feature. Then, the activation level of each feature would be given by:

$$a_k = \frac{1}{1 + e^{-\sum_i w_{ik} a_i}}, where\ w_{ii} = 0$$

Further massaging our derived expression for a_k gives:

$$a_k = \frac{1}{1 + e^{-\sum_i \frac{w_{ik}}{2} - \sum_i \frac{w_{ik}}{4} \sum_j w_{ij} a_j}} = \frac{1}{1 + e^{-\varphi - \sum_i \frac{w_{ik}}{4} \sum_j w_{ij} a_j}} \propto \frac{1}{1 + e^{-\varphi} e^{-\sum_i w_{ik}^2 a_j}}$$

where $\varphi = \sum_i \frac{w_{ik}}{2}$, a constant

Examining the last term in this expression reveals that this procedure is very similar to instantiating additional connections from each input node to every other input node whose strength is determined by and fixed to the strength of the connections between input layer and output layer, to a constant factor with weights approximately squared. This is close to the behavior I wished to emulate in the spatial excitement paradigm, as it is these connections that will allow me to activate the non-shared features of competitor nodes appropriately. The weights of these connections are already found for us, as a result of the inferred rules I calculate in the *exciteSecondary* procedure. I therefore use this procedure to emulate semantic dependence between features derived from their word-set memberships, allowing me to involve secondary features that were not

originally “in play” without changing the learning rule, in order to activate non-shared features of competitor outputs.

It should be noted that there is no reason this procedure could not be repeated multiple times. However, it can be shown that repetition of this procedure produces negligible changes in the weights very quickly. Each repetition doubles the exponent on the weight propagations. Since these weights are between 0 and 1, these repetitions will exponentially quickly produce weight changes approaching 0. Thus, my implementation uses a single application of this procedure in the modified network.

4.3.4: Limitations of the modification

Because the *activateSecondary* procedure partially emulates connectivity within the input layer, it also violates some assumptions of the Widrow-Hoff learning rule. As discussed, the learning rule as implemented requires the correct calculation of the direction of the steepest gradient from its current location in the network’s error-space. In order to calculate this gradient, it needs to calculate what the effects of connection weight changes will be. Because of the extra *activateSecondary* step, I violate the predictive power of the learning rule, which in turn no longer guarantees that it will converge directly to the nearest minimum. Fortunately for us, empirical testing of the modified network shows that it does eventually converge to this minimum, i.e. that the errors introduced by the *activateSecondary* method are not great enough to cause divergent behavior Unfortunately for us, as I have previously stressed, I am concerned with the evolution of these networks *over time* – in order to make useful comparisons between two

simulation runs, the networks must have had similar behaviors in approaching the 100% accuracy region.

The modified network is *not guaranteed* to approach the local minimum directly. Indeed, we see for many starting positions it often orbits around the local minimum, taking longer than expected by the gradient descent algorithm to reach it. Shown below is an example of this behavior demonstrated by the learning curve of a sample run on a simulation known to avoid the local minima:

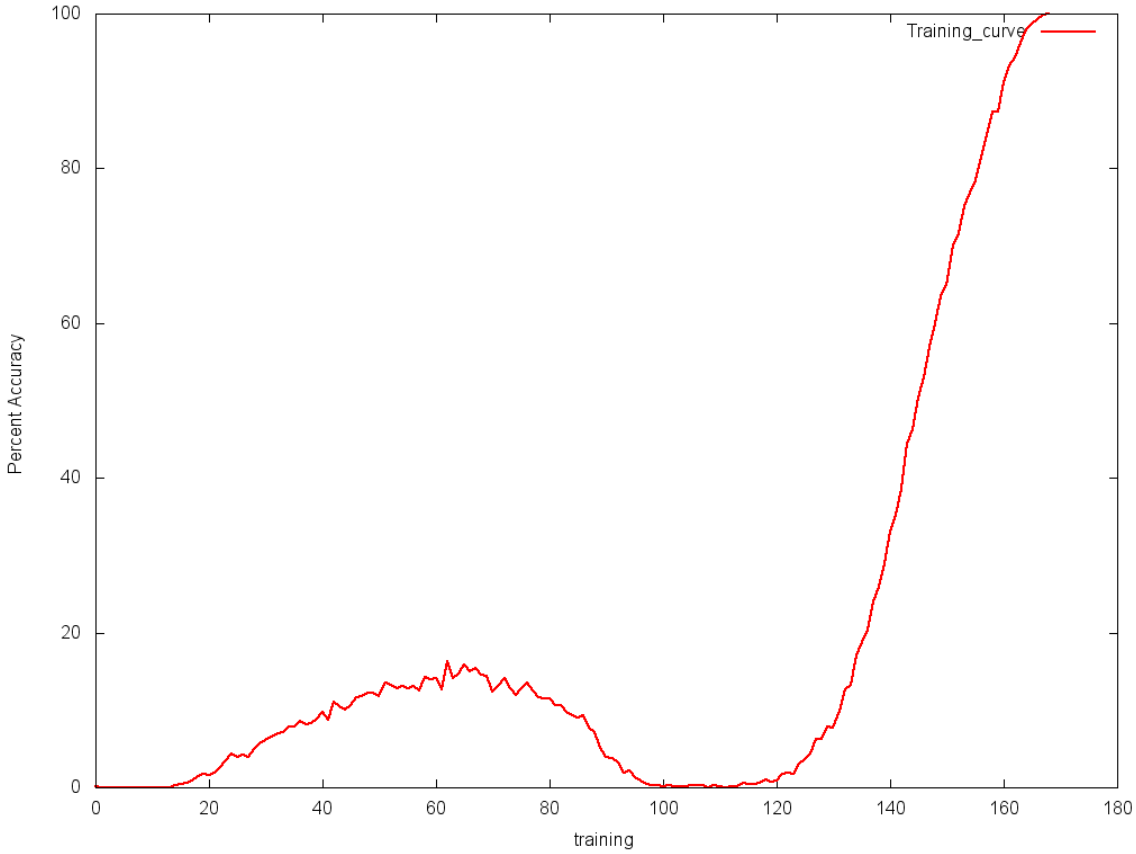

Figure 3 - Nonmonotic training curve produced by gradient descent with incorrect assumptions (taken from Simulation Group 1, 7 Shared Features, 16 Objects, 5 Shared Features, 0 Cross Features)

Unfortunately, lowering the learning rate does not solve this problem in all (or even most) cases. The only solution is start the gradient descent algorithm (i.e. initialize the connection weights) at a portion of the error-space that *happens* to proceed in the correct direction immediately. Because artificially calculating these locations in many ways begs the question (i.e. depends on external sources to solve the network rather than the network itself) I chose instead to discard data points generated by the modified network that do not monotonically approach the local minimum of the error-space *for the sake of analysis*. Please note that these networks still produce interference, and still correctly learn – they are just impossible to compare to the networks that immediately approach their respective local minima, as they take orders of magnitude longer to converge and produce very different boost counts (but that are still compatible with the requirements for semantic interference).

5: Empirical Evaluations

5.1: General Methodology

5.1.1: Overview

All evaluation of both the extended network (see Section 4.2) and the modified network (see Section 4.3) was carried out by simulating variations of experiments from the blocked-cyclic naming paradigm, described in Section 3.4. I considered a particular architecture as successfully modeling cumulative semantic interference if it was both (1) capable of stably learning (i.e. achieving 100% accuracy) the entire space of words presented to it in the training stage, and (2) capable of producing boosting curves for both the homogeneous and heterogeneous conditions which demonstrate both repetition priming and the effects of semantic interference where applicable. In these simulations, I expect both the homogeneous and heterogeneous conditions to demonstrate repetition priming, which will manifest itself as a reduction in boost counts as I repeat blocks. I expect to see semantic interference in the homogeneous conditions only. This will manifest as a steady increase in boost counts within individual blocks.

I will show that both proposed extensions to the original Oppenheim *et al.* model are successful in reproducing the expected effects. Once this is established, I will explore the parameter space of the simulated experiments in order to determine the effects of the internal structure of the dataset used for a given experiment. Because these relationships are generally difficult to control for real experiments, these simulations should offer some

quantitative insight into the expected strength of semantic interference as a function of the number of shared features per group, the number of groups, and other parameters.

The simulation results are broadly organized into three groups, with each group becoming progressively more general. In the final group, I use a subset of the McRae *et al.* (2005) norms for a number of experiments in order to show that both networks can scale to larger datasets. I also present a short section exploring the effect of noise on each network architecture.

5.1.2: Network Parameters

Unless otherwise specified, the network parameters for each simulation were set as follows:

Parameter	Value
Learning rate (η)	0.75
Activation noise (θ)	0.03
Boosting rate (β)	1.06
Threshold (τ)	1
Smoothing (σ)	100

Table 2 - Default Network Parameters

These values were chosen both to provide a reasonably large range for the boost outputs and to minimize the training time of the network. The smoothing parameter controls the number of times each simulation is run. With its value set to 100, each simulation presented in this thesis was run 100 times; their results were then averaged to produce the final output graphs. This effectively allows us to "smooth out" any

aberrations caused by noise, allowing us to see the results more clearly than a single run would allow.

As previously mentioned in Chapter 4, for my purposes, the number of training cycles used for each simulation cannot be simply fixed at 100 as was done in Oppenheim *et al.* Because I want to directly compare results between simulations of different experiments, I need to ensure that the networks involved in each of these simulations have been trained analogously to one another. To illustrate this point, consider two networks learning the same experimental dataset, one trained for 100 cycles and another trained for 1000. Clearly, the network trained for 1000 cycles will on average produce lower boost values for an identical simulation than the network trained for 100 cycles – it has had more time (in the form of additional training cycles) to further differentiate each output, thus lowering the boost count. Furthermore, consider two networks learning different datasets, one large and one small – if I train each network for 100 cycles, it is conceivable that the network operating on the smaller dataset will have achieved 100% accuracy while the network operating on the larger dataset will still make occasional errors – clearly the outputs of these two networks cannot be directly compared.

Thus, I establish the following convention: for all simulations presented, each network has been trained precisely the number of epochs required to reach 100% accuracy, and then immediately tested. This presents two opportunities: because I know each network has *just* reached 100% accuracy, I can compare their outputs, *and* I can use the number of epochs until 100% accuracy as a metric for evaluating how "difficult" a

particular dataset is to learn for the network, i.e. for estimating the time complexity of a given network as a function of the complexity and size of the dataset

5.1.3: Simulation and Dataset Parameters

For most of the simulations presented, the above network parameters are fixed. I instead vary the *simulation and dataset parameters* to try and evaluate the two networks performance over a wide variety of datasets. A list of simulation parameters that were varied and an approximate range over which they were varied is given below:

Parameter	Determined by	Range
Total no. of words (i.e. no. of output units)	Dataset	16-48
Total no. of features (i.e. no. of input units)	Dataset	14-274
No. of features per word	Dataset	2-21
No. of words per group	Simulation	4
No. of blocks	Simulation	6
Average no. of features shared between all members of a group	Both	1-7
Average no. of features shared between all members of multiple groups	Both	0-6

Table 3 - Simulation and Dataset Parameter Ranges

A number of the parameters above are determined solely by the structure of the data over which the simulation runs. These parameters include the overall size of the dataset, and the number of features required to specify a particular word. In order to control these parameters directly, I construct synthetic datasets with specific properties

for simulation groups 1 and 2. For simulation group 3, I leave these parameters to be determined by the implicit structure of the McRae *et al.* feature norm dataset

Two of the above parameters are controlled solely by the simulation setup. I fix the number of words per group at 4 as a matter of convention. Generally, blocked-cyclic naming paradigm experiments tend to set the group size at 4 as well. I also fix the number of blocks to 6, resulting in a 24-trial simulation length.

The final two parameters are determined by both the simulation setup and the underlying data: the inter- and intra-relatedness of any clusters present in the data. The average number of features shared within and across groups is determined by both the structure of the data and by the way in which I construct the particular groups for a simulation. In the next section, I introduce a metric for quantitatively measuring these values, along with a set a metrics for summarizing the output of a given simulation.

5.1.4: Metrics Used

I define a number metrics used to describe both simulation outputs and dataset structure. The first of these metrics is a measure of word dissimilarity. It is a function that takes two words and returns a value in the range (0.0, 1.0) which represents the amount of feature overlap that the two words have. If the value of the dissimilarity metric is 1.0, the two words have no features in common. If the value of the metric is 0.0, the two words are identical, and thus share all features and activation levels of those features. The metric is simply defined as the normalized Euclidean distance between the two words w_a and w_b in feature space as follows:

$$D_W(w_a, w_b) = \sqrt{\frac{\sum_k (f_{k_a} - f_{k_b})^2}{2}}$$

where f_{k_a} is the activation level (after applying the normalization routine outlined in section 4.2.2) of the k-th feature of word a and f_{k_b} is the activation level of the k-th feature of word b

The division by two is just to scale the outputted range of values from $(0, \sqrt{2})$ to (0, 1), as two completely orthogonal word vectors will be separated by a distance of $\sqrt{2}$ (they are all unit length due to the normalization routine).

This metric is useful for finding homogeneous groups in a large word-space. Simply test words pairwise until a clique of words with low average dissimilarity is found – this is a homogeneous group. I generalize this notion by defining a measure of *group* dissimilarity between G_A and G_B, where a group is a collection of words, as follows:

$$D_G(G_A, G_B) = \frac{\sum_i \sum_j D_W(G_{A_i}, G_{B_j})}{|G_A||G_B|}$$

where G_{A_i} is the i-th word of group A and G_{B_j} is the j-th word of group B, and $|G|$ is the number of elements in group G.

This metric operates identically to the word dissimilarity metric, but for groups of words instead of individual words. A group dissimilarity of 1.0 means that the two groups in question share no features, while a group dissimilarity of 0.0 means that there is a bijective mapping between the two groups such that each word and its image are identical.

Using this definition for group-dissimilarity, I can define a third useful metric for determining the heterogeneity of a given group: auto-dissimilarity. The auto-dissimilarity of a group G is given by:

$$D_A(G) = D_G(G, G)$$

I say that a group G is *more heterogeneous* than a group H if $D_A(G) > D_A(H)$.

The metrics above are useful for examining the internal structure of a given dataset, for finding homogeneous groups within that dataset, and for determining the relationships between groups once they are chosen.

After a set of groups is chosen for a given experiment, I execute the simulation. I define a number of metrics over the outputs of these simulations for summarizing and comparing results between a large number of simulations.

The first of these metrics is the training period T, which I define as the number of epochs required to reach 100% accuracy. I seek to show, as expected, that this metric is generally a function of the overall size of the dataset.

The second metric seeks to quantify approximately how *much* semantic interference is occurring for a given data set. It is defined as follows:

$$\bar{\mu} = \frac{|\overline{m_{heterog}}|}{|\overline{m_{homog}}|}$$

where $\overline{m_{heterog}}$ is the average slope of the heterogeneous groups boost counts over trials and $\overline{m_{homog}}$ is the average slope of the homogeneous groups boost counts over trials

Because semantic interference is observed via an increase in boost counts within a block, and this interference should only occur for homogeneous groups, heterogeneous groups should, on average, have a steeper slope than their homogeneous counterparts. This

metric simply calculates the ratio of the slopes between the two conditions – values larger than 1 indicate an interference effect, with larger values indicating more interference. I seek to show that this metric is a function of the dissimilarity metrics presented earlier. This would imply that the magnitude of semantic interference observed is a function of the homogeneity of the dataset, as would be expected from a system that claims to model this effect.

5.1.5: Implementation Details

Once the network is trained for its training period (T), the simulation produces multiple copies of the network. Each copy then simulates a particular condition's block-cycle as would be expected. This entails presenting the set of words in that particular condition in random order (over the course of a single block) for the specified number of cycles. The results from each copy are then collated into a single graph. In this way, I prevent the ordering of the condition presentations from affecting the network's output. As previously mentioned, these steps are repeated σ times and averaged to produce the final output graphs.

The construction of heterogeneous conditions proceeds as in real experiments – a single member from each homogeneous group is chosen and combined to create a condition that is guaranteed to be heterogeneous with respect to the homogeneous conditions. Multiple heterogeneous conditions can be constructed this way – indeed, the number of possible heterogeneous conditions able to be constructed from N sets of M elements is given by:

$$|H| = M^N$$

For the simulations, I use two different, randomly generated heterogeneous conditions, constructed from the set of homogeneous conditions as just described.

5.2: Simulations

5.2.1: Showing Semantic Interference

Before the results of the simulations from the three groups are presented, it is first important to establish that both the extended and the modified networks produce the expected semantic interference from the baseline network, presented in Section 4.1.2. Some summary parameters of the network are given below:

Parameter	Value
Total no. of words (i.e. no. of output units)	16
Total no. of features (i.e. no. of input units)	20
No. of features per word	2
No. of features shared between all members of a group	1

Table 4 - Summary Parameters of the Baseline Experiment

Because both the extended and the modified networks allow for variable activation levels of the input features for a given word, each word was defined to equally weight both of its constituent features in order to conform to the binary activation levels present in the original network. The results of this simulation on both networks are presented below, plotted as the selection time in boosts as a function of the trial number:

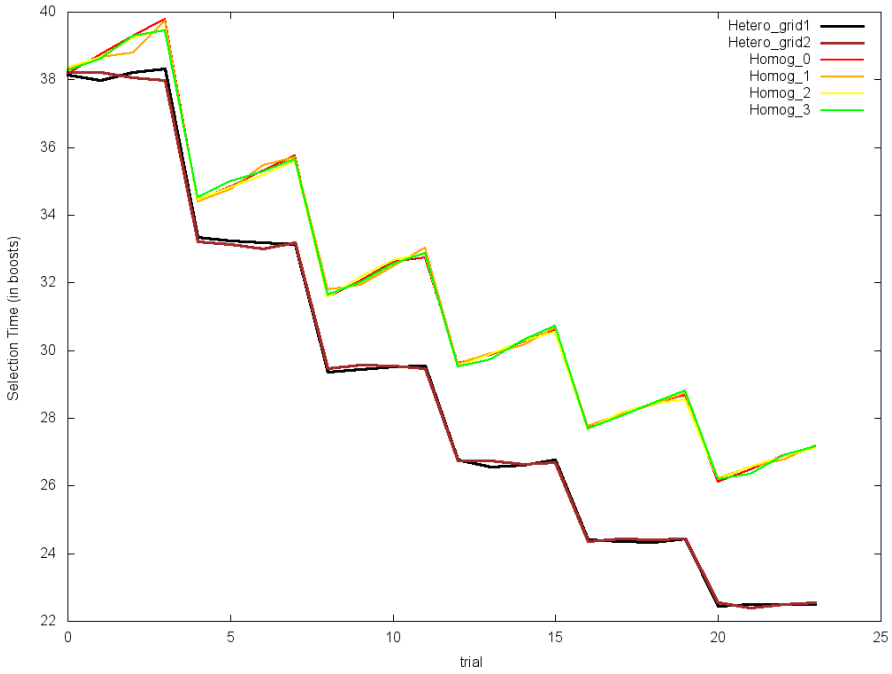

Figure 4 - Baseline Simulation, Extended Network

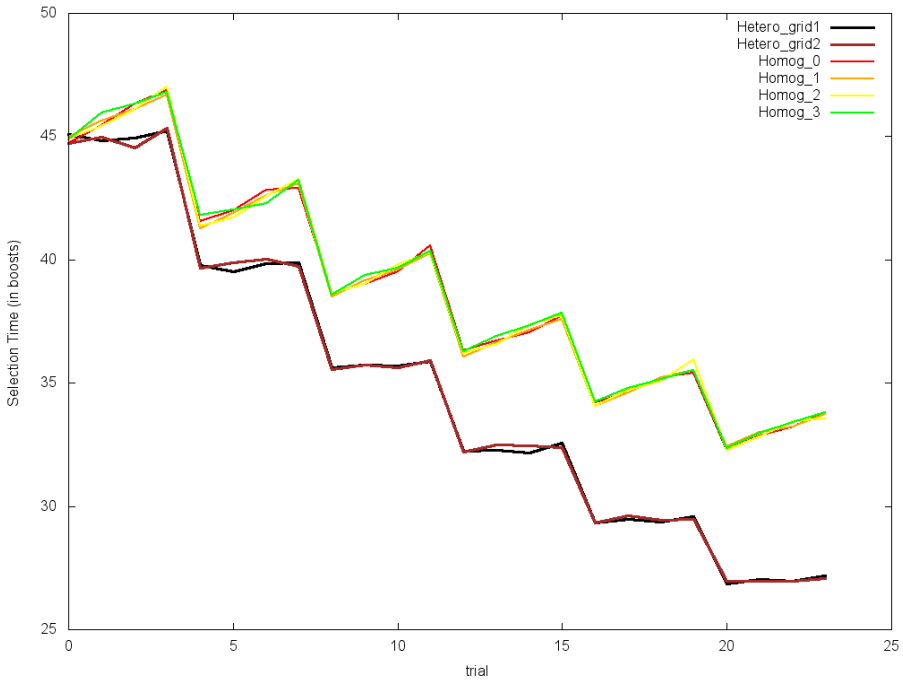

Figure 5 - Baseline Simulation, Modified Network

Note that the intra-block boost counts for the homogeneous conditions in both graphs increase, while the intra-block boost counts for the heterogeneous conditions remain constant. This is indicative of semantic interference effects. Also note the overall inter-block boost count improvements in both graphs for all conditions. This is indicative of repetition priming effects. Taken together, we have strong evidence for cumulative semantic interference effects in *both* networks. Indeed, both networks perform identically on this simulation barring the relative difference in boost counts. Thus, both networks successfully reproduce the results of Oppenheim *et al.* Now, I present three groups of simulations, each of increasing internal complexity, that seek to generalize these results to larger and more complex networks.

5.2.2: Simulation Group 1

Simulation Group 1 was designed in order to explore direct generalizations of the baseline simulation while deviating as little as possible from the limitations set forth in the original model. Because of this, Group 1 is the *least* general of the 3 simulation groups, and thus explores a very small subspace of the full network and simulation parameter spaces. However, because I limit the parameter space so severely, I am able to fully cover significant portions of it via simulation, allowing for nearly exhaustive testing of the subspace.

In Simulation Group 1, each simulation's homogeneous conditions have identical structure. For example, if there are 4 groups in a given simulation, each of these 4 groups will consist of 4 words, which will each share the same number of features between them.

Furthermore, the activation levels of the features corresponding to every word in the simulations in Group 1 are equal, in accordance with the baseline simulation.

In Simulation Group 1, I varied the simulation parameters according to the table below:

Number of Features per Object	Number of Homogeneous Groups (G_{count})	Number of Shared Features within each group (f_{shared})	Number of Shared Features across each group (f_{cross})
2	4	1	0
3	6	2	1
4	8	3	2
5	12	4	3
6		5	4
7		6	5

Table 5 - Parameter Values for Simulation Group 1

Every possible combination of each of these parameters was simulated. I discard logically inconsistent combinations of the above parameters and further stipulate that each word must have at least one unique feature, in order to avoid degenerate cases with identical words. After these combinations are removed, we are left with a grand total of 224 simulations. Each of these simulations was run on both network architectures. The metrics discussed earlier in this chapter were then calculated and combined into summary graphs.

I first look at the case where the number of shared features across groups (hereafter referred to as “cross features”) is 0 – this corresponds exactly to the baseline model, which had 1 shared feature and 1 unique feature for every word. By fixing this parameter, we can visualize the residual four dimensional space in 4 three dimensional slices, each corresponding to a different value for the number of homogeneous groups.

Presented below are 2 of those slices – the highest and lowest, corresponding to 4 groups and 12 groups respectively:

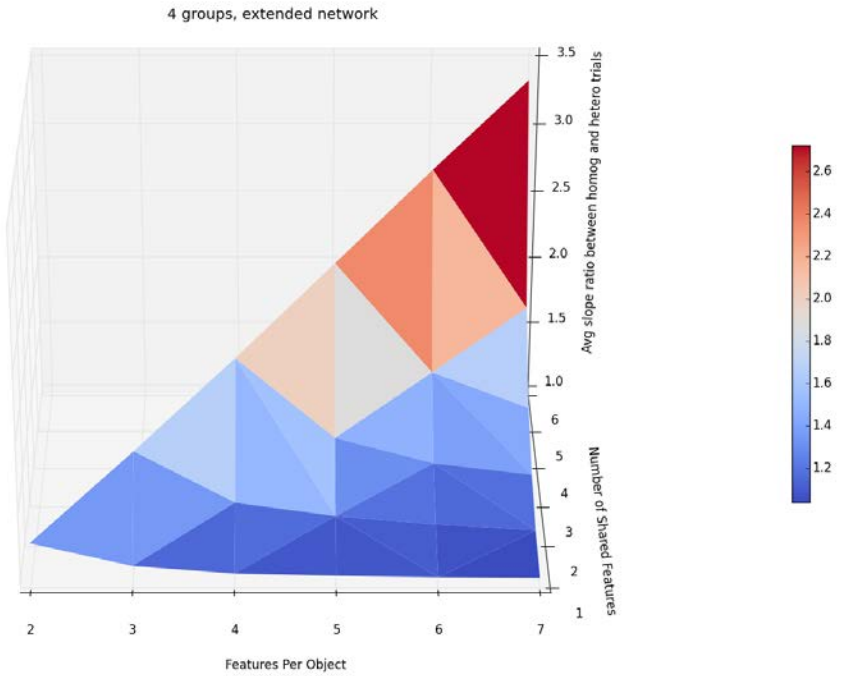

Figure 6 - $\bar{\mu}$ as a function of Shared Features and Features per Object in the Extended Network with no cross features, 4 groups

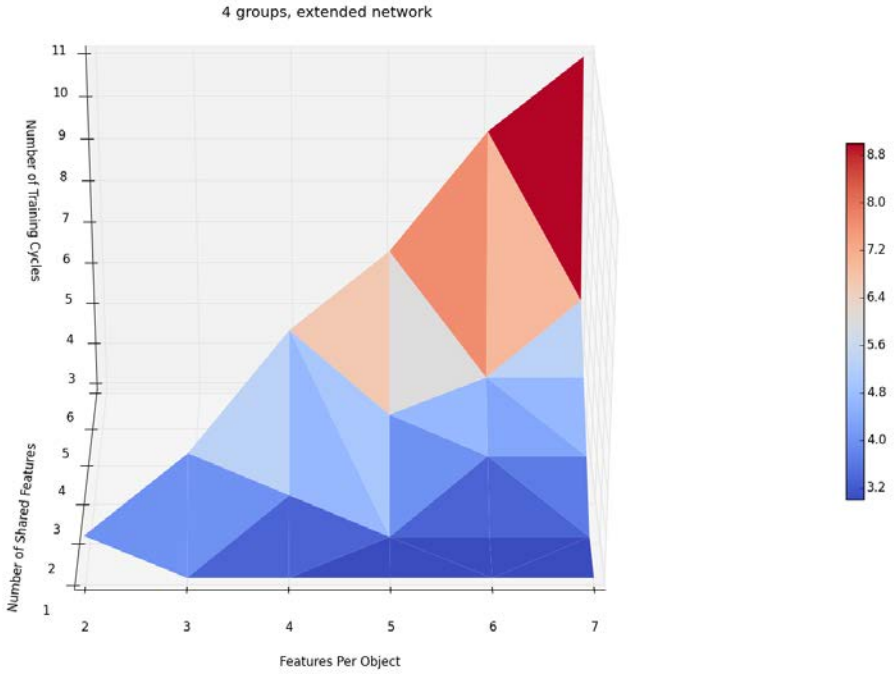

Figure 7 - T as a function of Shared Features and Features per Object in the Extended Network with no cross features, 4 groups

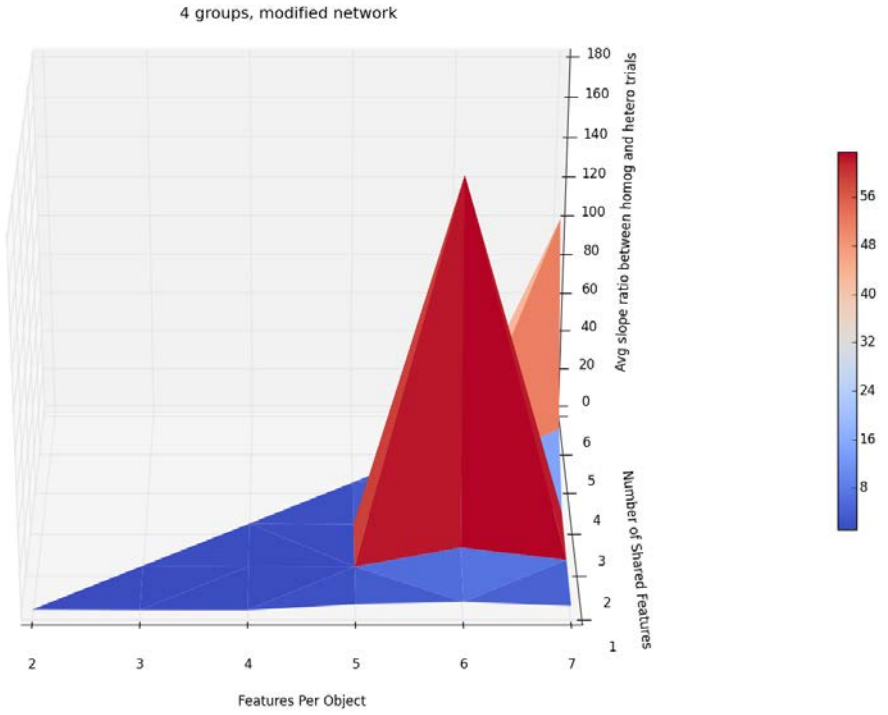

Figure 8 - $\bar{\mu}$ as a function of Shared Features and Features per Object in the Modified Network with no cross features, 4 groups

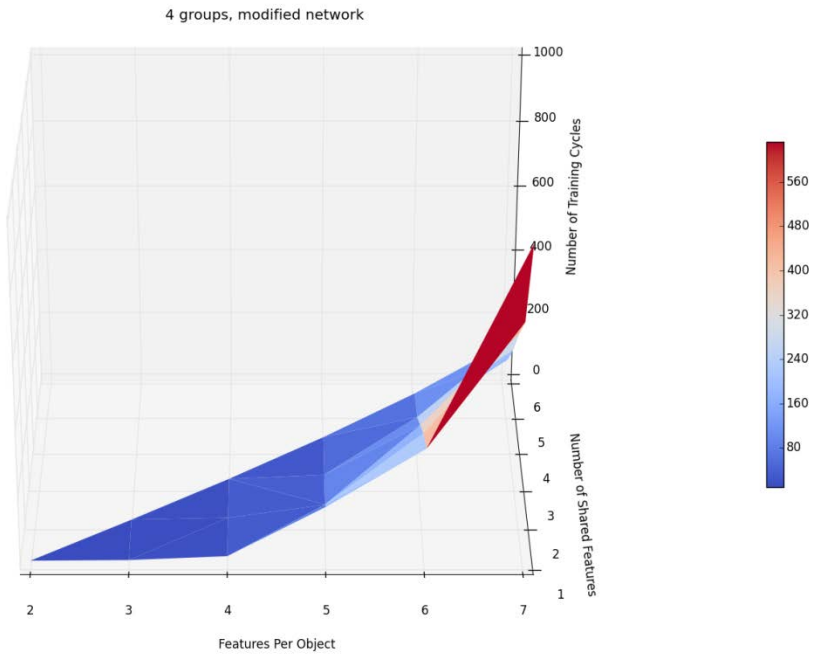

Figure 9 – T as a function of Shared Features and Features per Object in the Modified Network with no cross features, 4 groups

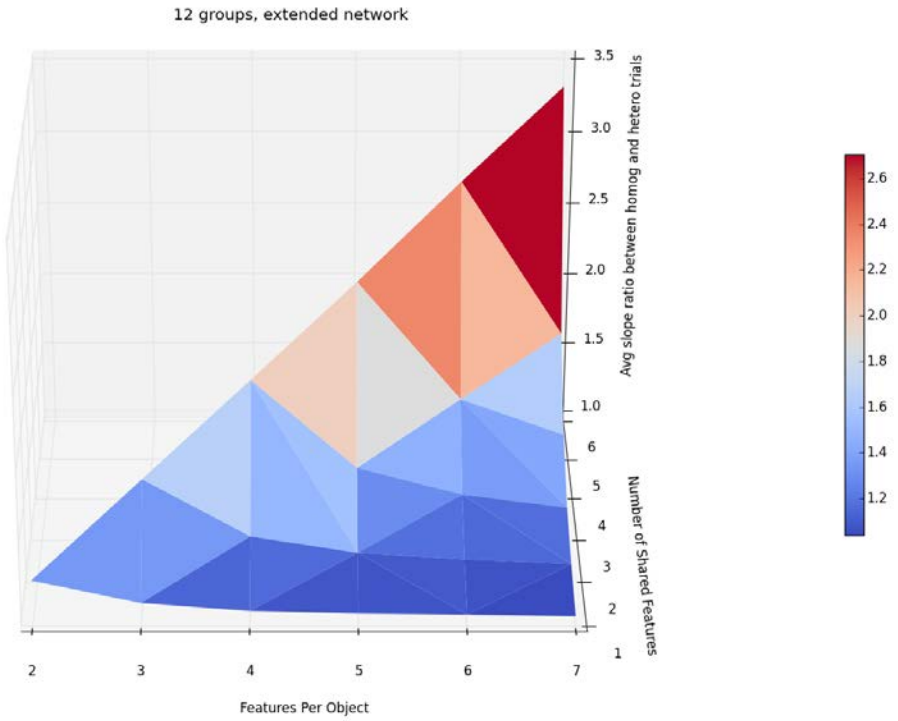

Figure 10 - $\bar{\mu}$ as a function of Shared Features and Features per Object in the Extended Network with no cross features, 12 groups

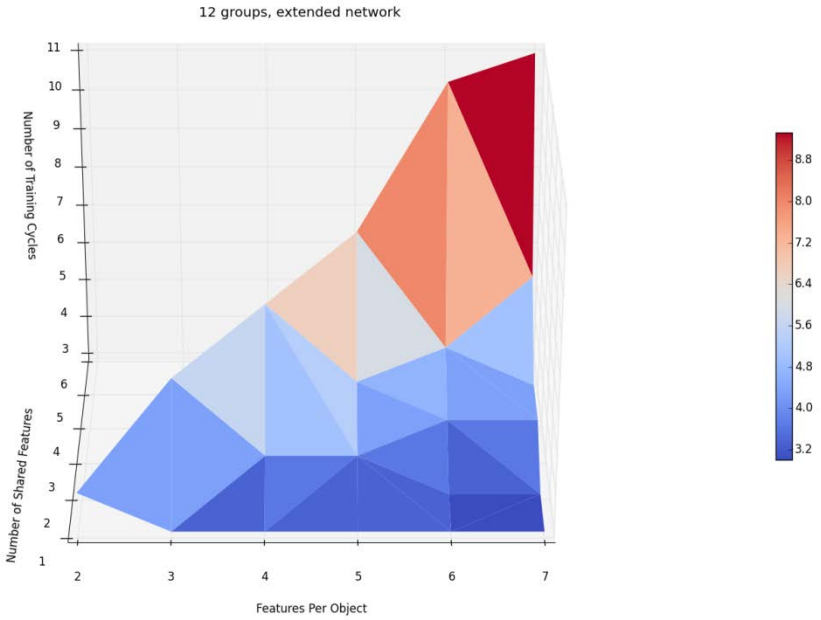

Figure 11 - T as a function of Shared Features and Features per Object in the Extended Network with no cross features, 12 groups

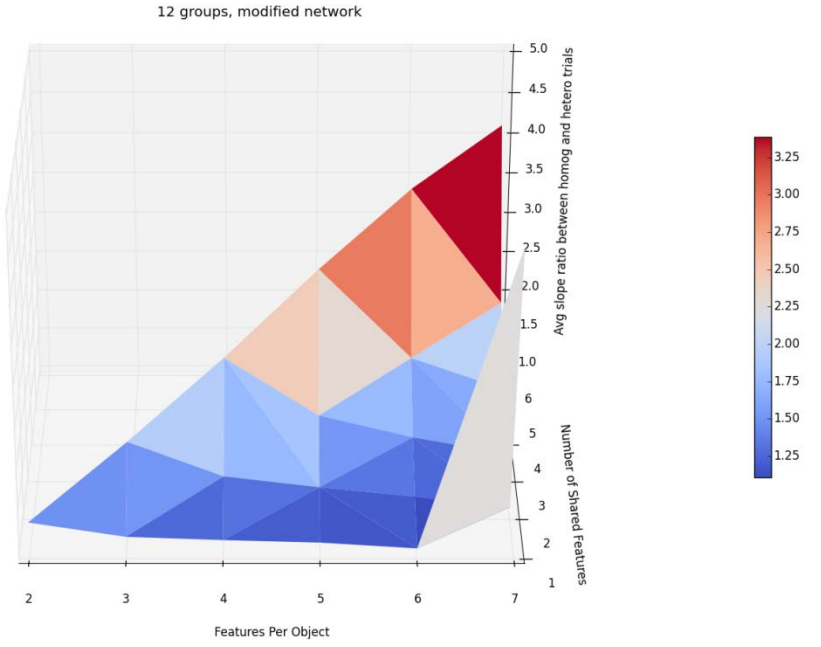

Figure 12 - $\bar{\mu}$ as a function of Shared Features and Features per Object in the Modified Network with no cross features, 12 groups

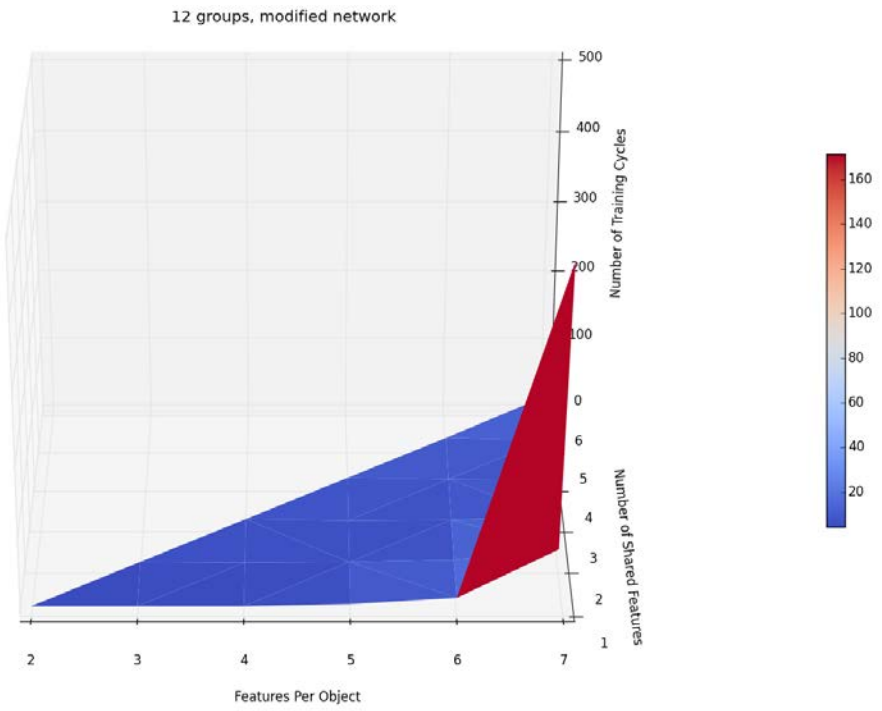

Figure 13 - T as a function of Shared Features and Features per Object in the Modified Network with no cross features, 12 groups

Examining the extended network's outputs allows us to draw some early conclusions about the effects of network size and group composition on both $\bar{\mu}$ and T.

Note the similarities between Figures 6 and 10 – even though Figure 10 shows a network 4 times as large, the training periods for each simulation remained unchanged. This makes sense when one considers that all inputs are trained in parallel during a particular epoch. As we grow this particular simulation from the 4 group to the 12 group case, no interdependence exists between the original four groups and the additional groups added; thus, we see no increase in training period for the larger network. We will see this size invariance no longer holds as the interdependence between groups is increased by introducing features shared *across* groups.

Comparing the $\bar{\mu}$ graphs (figures 5 and 9) for the extended network, we see identical shapes. Because the overall structure of the dataset is not being modified when the size of these simulations is increased, this is the expected result. If the groups shared any features between each other, however, this size invariance would no longer hold true, as we will see later.

Finally, we see very clearly the effect of group composition on both $\bar{\mu}$ and T. In both cases, we see that *the amount of interference observed varies as a function of the ratio of shared features to total features*:

$$\bar{\mu} \sim \frac{f_{shared}}{f_{total}}$$

However, we know that this ratio is itself proportional to one of the previously defined metrics, the auto-dissimilarity, $D_A(G)$. Because all the groups in these simulations are identical, we can typify each simulation by a single auto-dissimilarity value, given by:

$$D_A(G) = D_G(G,G) = \frac{\sum_i \sum_j D_W(G_i, G_j)}{|G|^2} = \frac{12 * D_W(G_0, G_1)}{16} = \frac{3}{4}\sqrt{\frac{(f_{total} - f_{shared})}{f_{total}}}$$

$$= \frac{3}{4}\sqrt{1 - \frac{f_{shared}}{f_{total}}}$$

Furthermore, we can empirically relate this expression to the output values for $\bar{\mu}$ as:

$$\bar{\mu} \approx e^{\frac{.55}{D_A(G)} - .75} = \beta \exp\left(\frac{\propto}{\frac{3}{4}\sqrt{1 - \frac{f_{shared}}{f_{total}}}}\right) + \gamma$$

where $\propto$, β, and γ are arbitrary scaling constants and are dependent on the normalization routine chosen. The values given above work relatively well for the normalization routine (where we normalize every vector to length 1)

A similar expression can be derived for T.

Examining the modified network's outputs highlights the issues discussed in section 4.3.4. In the 4 group slice (Figure 7), the discontinuities make it very difficult to read the output strictly from the graph; however, manual examination of the data shows that removing these discontinuities results in a graph nearly identical to Figure 11 as expected. Removing the single discontinuity in Figure 11 (Features/Object = 7, Shared Features = 1) gives us a graph with the exact same shape as Figures 5 and 9 – in other words, the modified network produces the same analysis (when it can find the correct solution immediately) as the extended network. Furthermore, the modified network actually produces *more* semantic interference than the extended network for the same simulation configuration, i.e. its value for $\propto$ is higher for a given network configuration than the extended network. However, examining Figures 8 and 12 show that this increase in distinguishability comes at the cost of training time – the training period T for the modified network tends to be much larger than the training period for the extended network on the same simulation. These differences will become most obvious in Simulation Group 3, over the full McRae norms. This further implies that the scaling constant $\propto$ is operant in both the expression for $\bar{\mu}$ as well as the expression for T – indeed, in the simulation results we often see a correlation between the values for T and the values for $\bar{\mu}$. It is unclear if this relationship holds in real experiments – it may be the

case that more complex concepts that take longer to learn tend to produce more semantic interference in trials than simpler concepts. This will be further discussed in Chapter 6.

Thus far I have examined only cases wherein the individual groups are both identical and independent, making both T and $\bar{\mu}$ essentially independent of network size. In Simulation Group 1, I also varied the number of *cross features* present in the groups – features that are unilaterally shared across all groups. This still keeps the groups identical, but allows them to be dependent on one another in a very controllable way – I can (to an extent) control the group dissimilarity by varying the number of cross features in each group. Shown below are more 3-dimensional slices of the results. These are sliced along different dimensions, however – I fix the number of features per object (in this case, 7, in order to show the results at the highest resolution simulated) as well as the number of groups, leaving a three dimensional space with x-axis representing *shared features within groups* and y-axis representing *shared features across groups*. I present two slices, taken with $G_{count} = 4$ and $G_{count} = 12$ of the extended network below:

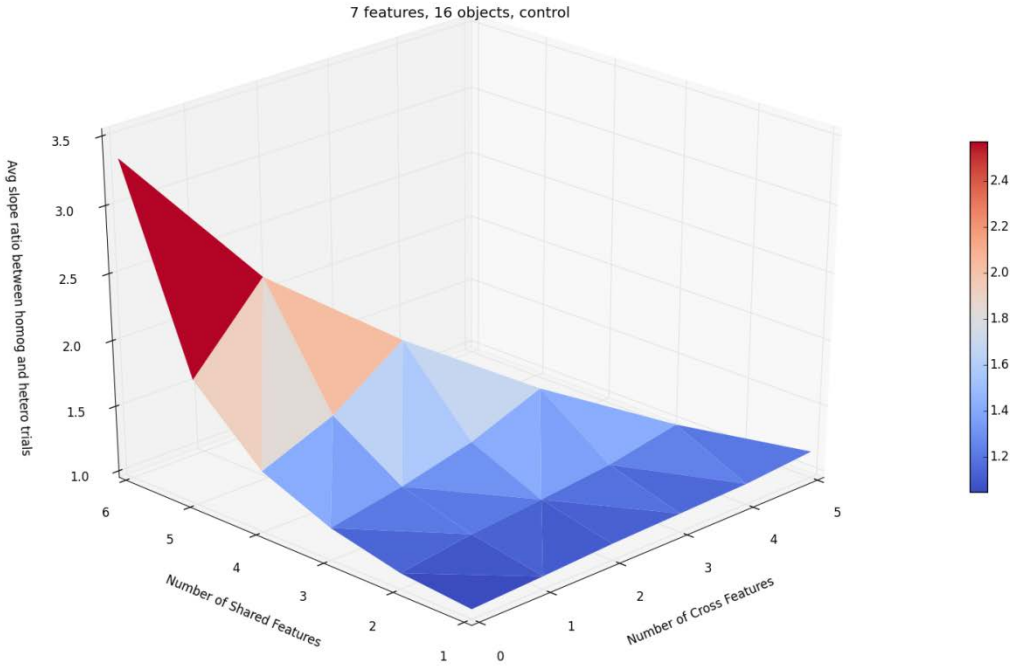

Figure 14 - $\bar{\mu}$ as a function of Shared Features and Cross Features in the Extended Network with 7 features per object, 4 groups

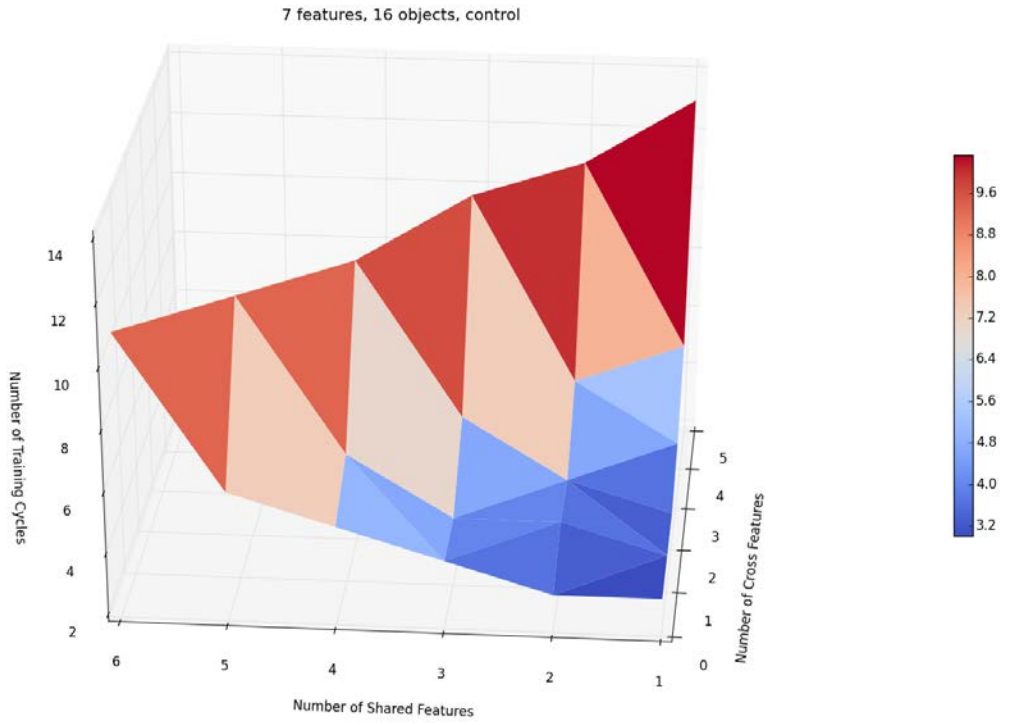

Figure 15 - T as a function of Shared Features and Cross Features in the Extended Network with 7 features per object, 4 groups

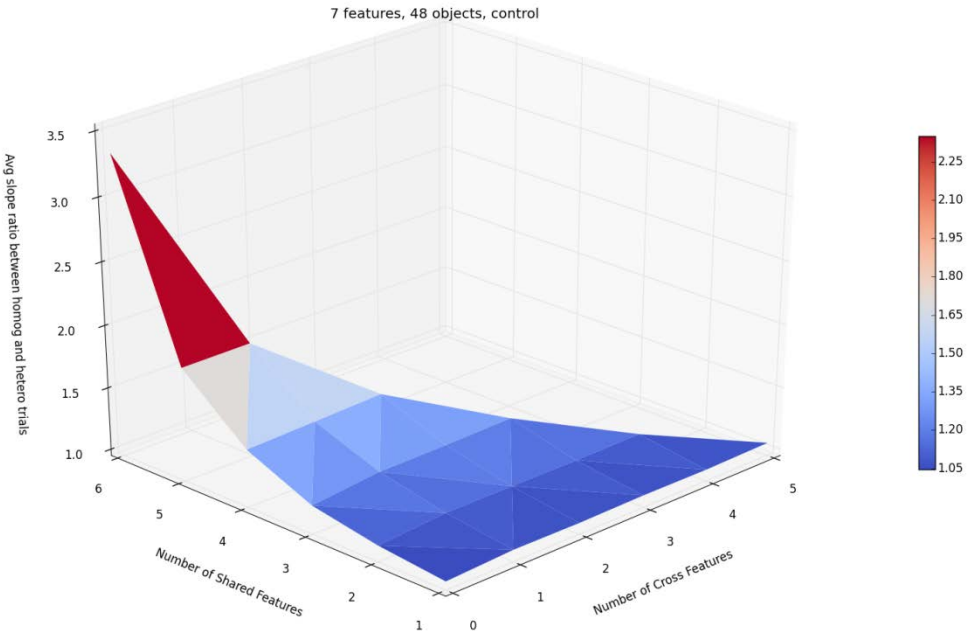

Figure 16 - $\bar{\mu}$ as a function of Shared Features and Cross Features in the Extended Network with 7 features per object, 12 groups

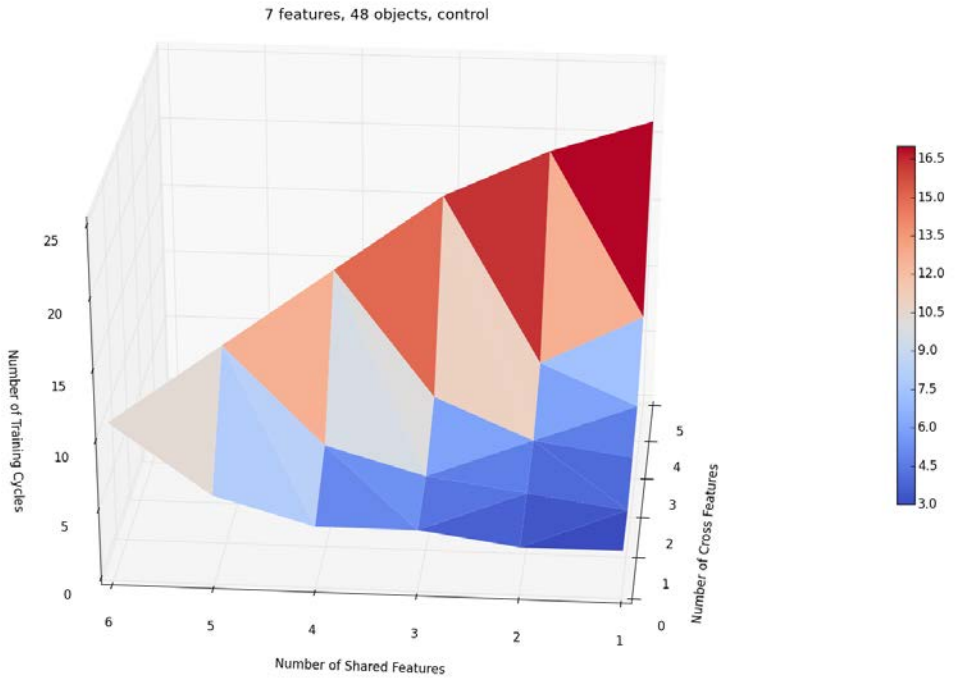

Figure 17 - T as a function of Shared Features and Cross Features in the Extended Network with 7 features per object, 12 groups

From just these two slices, we can see a lot of interesting results. First, as we expect, we see that T is no longer independent of the network size. While the 2-d dimensional slice along $f_{cross} = 0$ is identical in both cases, for the larger network the results for T are unilaterally larger than the smaller network's results. This is because with the addition of the cross features, the groups are no longer independent, and so affect each other during the training epochs – in this case negatively, requiring more epochs to reach 100% accuracy. Thus for more general trials we should expect more complex relationships between groups to yield longer training periods.

We also see that $\bar{\mu}$ tends to increase as the number of cross features increases. Furthermore, we see that it increases *faster* when the number of shared features is higher. This effect is actually quite easy to explain. In my earlier analysis, I disregarded the value of any group dissimilarities that were not auto-dissimilarities because the value of all non-auto-dissimilarities was always 1.0. When cross features are introduced, however, this assumption is no longer true; thus, we must amend the expression for predicting the amount of interference observed. Recall that $D_A(G)$ is a measure of how dissimilar a group is from itself, which is to say that if its value is lower, then we expect that particular group to contain elements that are highly semantically related to one another. Recall also that $D_G(G_1, G_2)$ gives the same measure across groups – if this measure is low, we expect these groups to be *difficult to semantically distinguish from one another*. When this is the case, the heterogeneous group constructed from them will *not operate sufficiently differently from a homogeneous group*. In other words, the lower $D_G(G_1, G_2)$ is on average, the more similar the heterogeneous control group must be to the

homogeneous groups. Since our estimate of interference depends on the assumption that the heterogeneous group is distinguishable from the homogeneous group, then $\overline{D_G}$ will tend to *suppress* interference effects, where $\overline{D_G}$ is the average pairwise group dissimilarity defined as:

$$\overline{D_G} = \frac{\sum_i \sum_j D_G(G_i, G_j)}{G_{count}(G_{count} - 1)}, i \neq j$$

where G_{count} is the number of groups being compared

For these simple networks, all the group dissimilarities are equal (when $i \neq j$), and thus

$$\overline{D_G} = D_G(G_1, G_2)$$

where G_1, G_2 are any two groups and $G_1 \neq G_2$

Thus, we can amend our old expression by introducing this new term as such:

$$\bar{\mu} \approx \beta \mathrm{e}^{\frac{\propto}{\frac{D_A(G)}{\overline{D_G}}}} + \gamma$$

In the larger network, however, we see that the contribution from the cross features seem negligible. In order to explain this effect, we must examine some specific data from the simulations. Shown below are the points ($f_{shared} = 5, f_{cross} = 0$) and ($f_{shared} = 5, f_{cross} = 1$) in the 4 group simulation as well as the 12 group simulation:

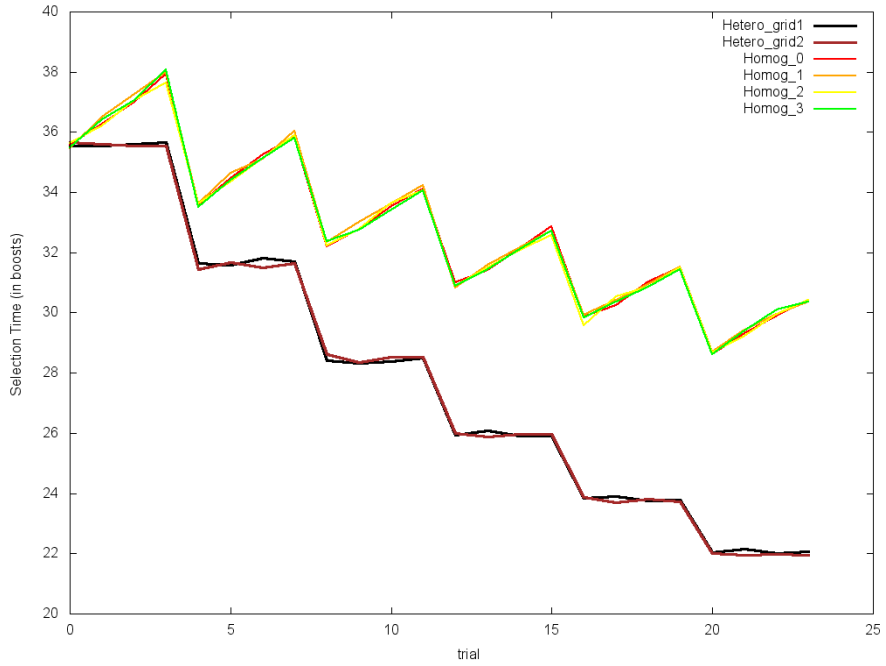

Figure 18 - Boost count output over time, 7 features per output, 5 shared features, 0 cross features, 4 group network

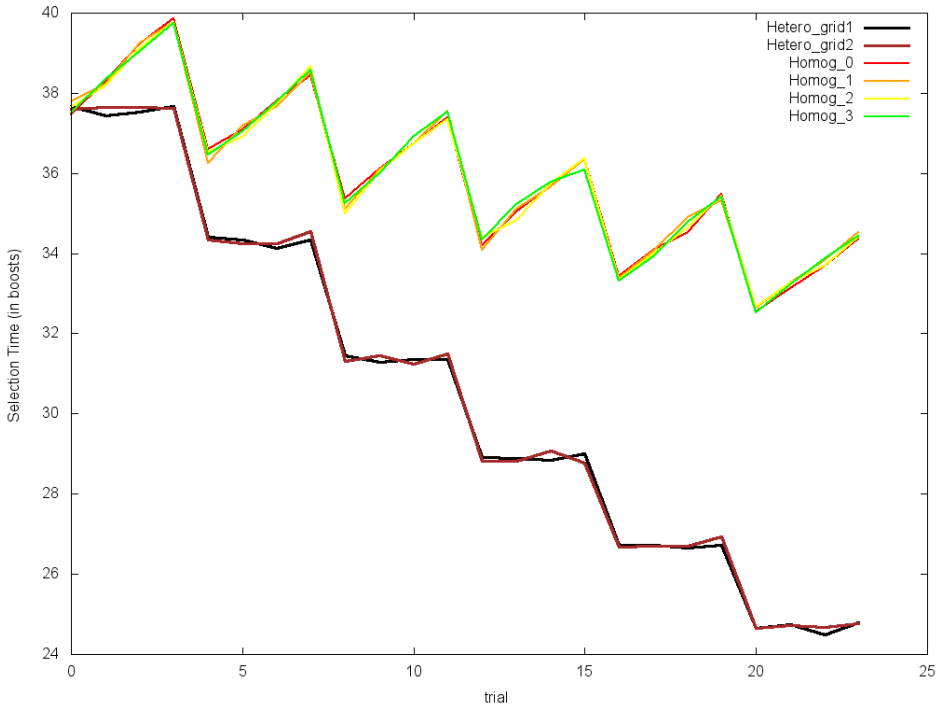

Figure 19 - Boost count output over time, 7 features per output, 5 shared features, 1 cross feature, 4 group network

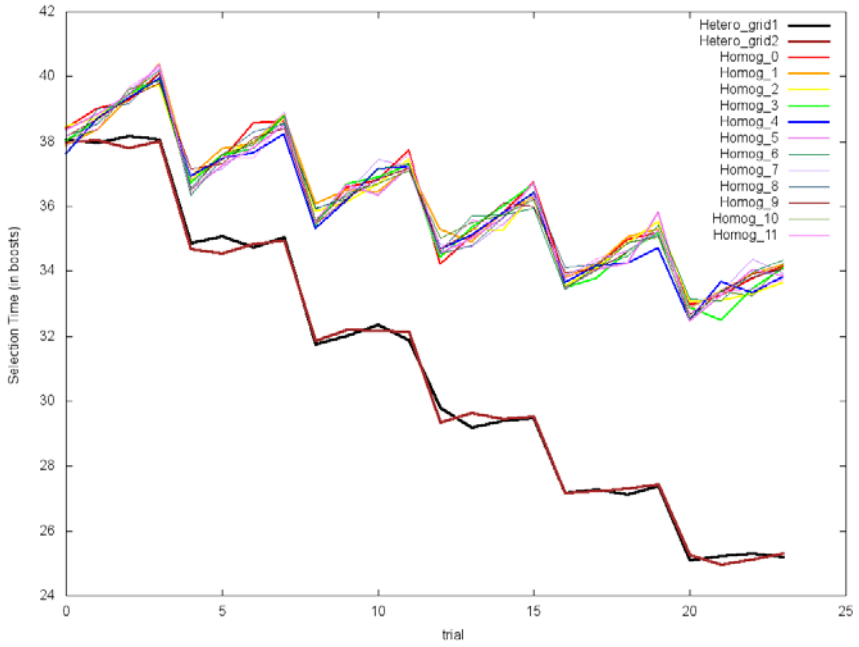

Figure 20 - Boost count output over time, 7 features per output, 5 shared features, 0 cross features, 12 group network

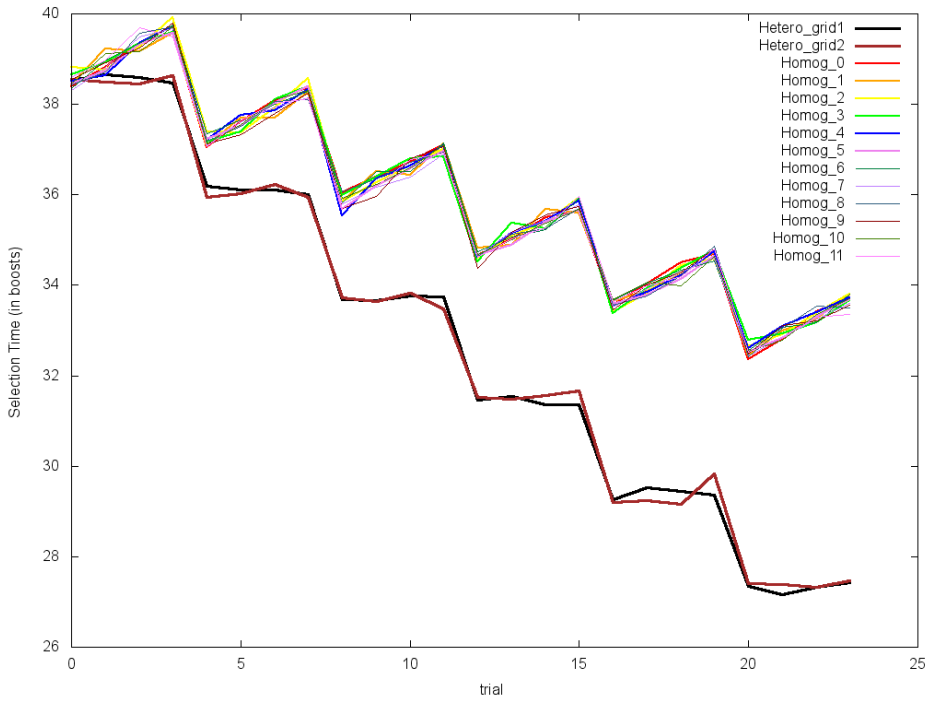

Figure 21 - Boost count output over time, 7 features per output, 5 shared features, 1 cross feature, 12 group network

Careful analysis of the above graphs shows that for both graphs in both network sizes, the slope of the line of best fit for the heterogeneous conditions remain relatively constant in both simulations. However, in the smaller network, the slope of the homogeneous conditions is significantly lower for the case where $f_{cross} = 0$. The overall slope of these conditions are a result of a combination of positive regions (regions wherein we are experiencing semantic interference) and negative regions (regions where we feel the effects of repetition priming). Measuring the slopes of each of these regions across both graphs finally reveals the culprit. In both cases, the regions with positive slope have $\bar{m} \approx 2$. However, the regions with negative slope (the regions caused by repetition priming effects) in the graph with $f_{cross} = 0$ are almost twice as steep as those in the graph with $f_{cross} = 1$. Thus, we can conclude that the discrepancies in the behavior of the two simulation sizes are due to repetition priming effects, which in turn affect our metric for estimating the magnitude of semantic interference occurring in the simulation. Why then, do we see invariant repetition priming effects in the larger network? This mostly has to do with the way in which I defined the training period. Because we stop as soon as 100% accuracy is reached, and because training epochs are necessarily quantized (you cannot stop halfway through a training period) the smaller network is "undertrained" in some sense, insofar as repetition priming effects have much greater magnitude due to the connection weights have more room for optimization left in them when the training period ends.

This should be encouraging to designers of experiments that wish to explore semantic interference effects. It essentially claims that when selecting homogeneous sets,

it is unimportant to consider features that they all might share across sets, as these features will tend to be inconsequential for large networks. Since features that are shared across sets tend to be the hardest to recognize, it is important to know that they do not affect any results in the limit.

Let us briefly consider the above simulations as interpreted by the modified network:

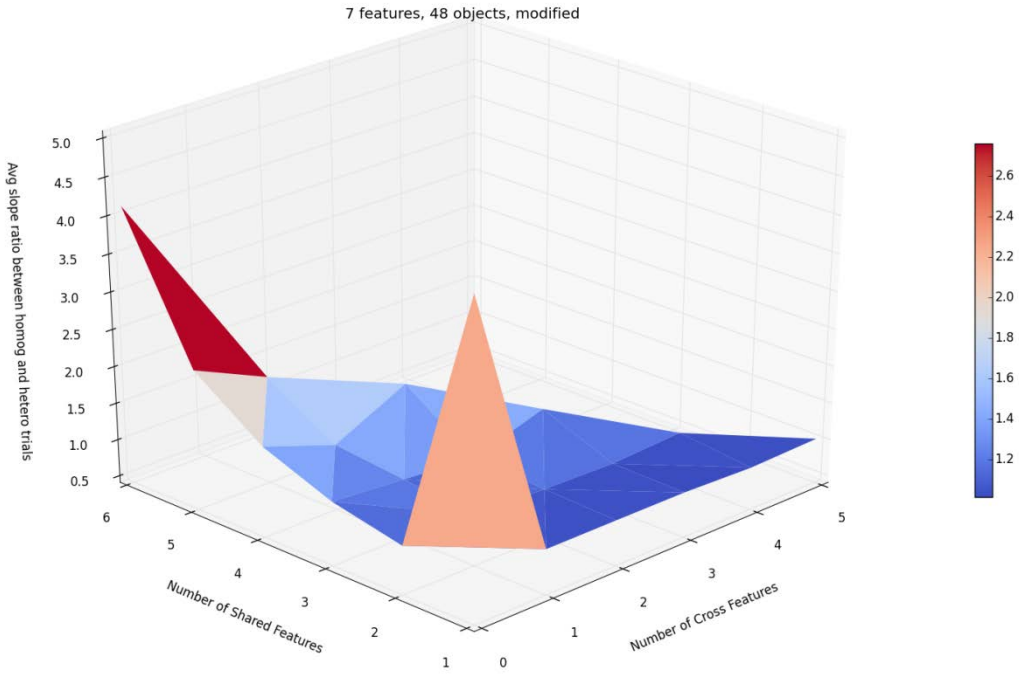

Figure 22 - $\bar{\mu}$ as a function of Shared Features and Cross Features in the Modified Network with 7 features per object, 12 groups

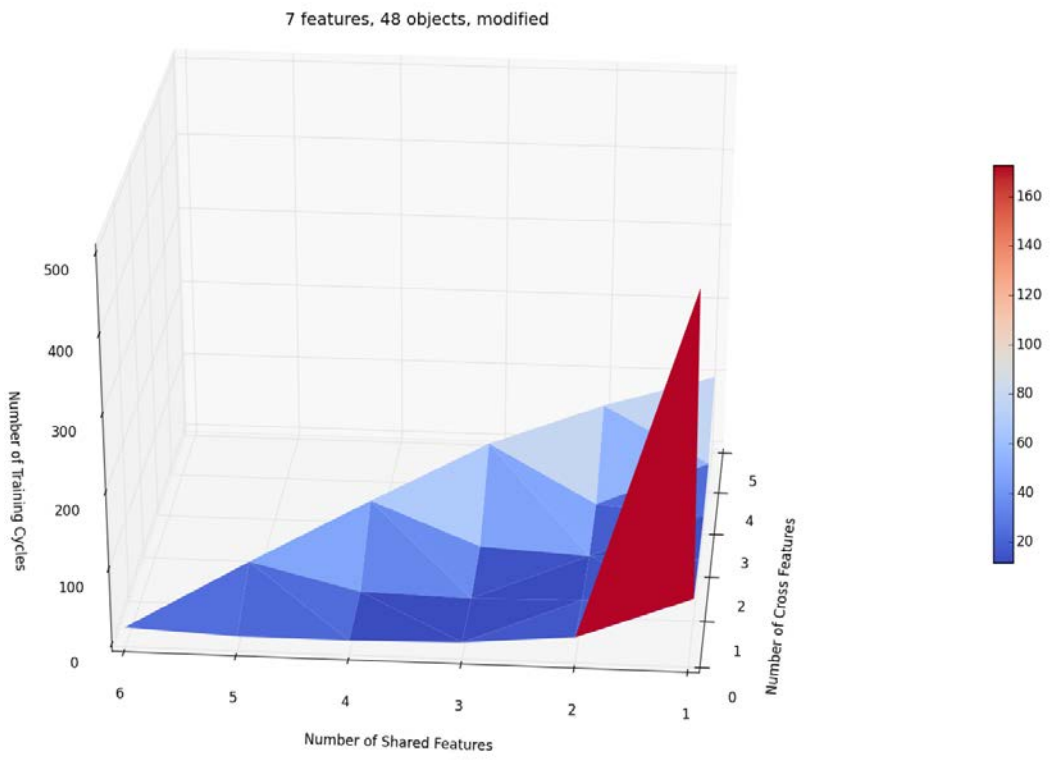

Figure 23 - T as a function of Shared Features and Cross Features in the Modified Network with 7 features per object, 12 groups

Removing the discontinuity at $(f_{shared} = 1, f_{cross} = 0)$ gives us an output nearly identical to the extended network, as expected.

5.2.3: Simulation Group 2

In Simulation Group 2, I begin to execute simulations that have different, independently defined groups. Furthermore, I begin to vary many of the parameters kept constant for Group 1. The simulations in Group 2 were designed to cover as much of the parameter space as possible. Exhaustive testing in this regime is no longer possible – there are simply too many combinations of parameters. Thus, I designed simulations designed to sample the portions of the parameter space that I felt were important or instructive. Shown below is a graph tabulating the simulations that comprise Simulation

Group 2. In this table, I describe the structure of each of the four homogeneous groups that comprise the particular experiment:

Simulation	Group 1 Features/Object	Shared features in Group 1	Group 2 Features/Object	Shared features in Group 2
2.1	3-7	1-2	5	2
2.2	3-7	2-3	5	2
2.3	5	2	3-5	2
2.4	5	2	3-7	2
2.5	3-7	2	3-5	2
2.6	3-7	2	3-7	2

Table 6 - Simulation Group 2 Summary, homogeneous groups 1 and 2

Simulation	Group 3 Features/Object	Shared features in Group 3	Group 4 Features/Object	Shared features in Group 4
2.1	5	2	5	2
2.2	5	2	5	2
2.3	4-6	2	5-7	2
2.4	3-7	2	3-7	2
2.5	4-6	2	5-7	2
2.6	3-7	2	3-7	2

Table 7 - Simulation Group 2 Summary, homogeneous groups 3 and 4

Furthermore, the weights of these features are no longer constant. I set the input activation levels by randomly sampling norms from the pool of the McRae *et al.* feature set, which gives us a realistic activation level distribution while still using synthetically constructed sets.

I will discuss each simulation individually. A summary of metrics calculated and observed at output for each simulation is also presented below:

Simulation	Group 1 D_A	Group 2 D_A	Group 3 D_A	Group 4 D_A	Heterogeneous Group D_A	$\overline{D_G}$
2.1	.639	.581	.581	.581	.75	1
2.2	.541	.581	.581	.581	.75	1
2.3	.581	.523	.566	.616	.75	1
2.4	.581	.542	.547	.616	.75	1
2.5	.576	.523	.566	.616	.75	1
2.6	.576	.542	.547	.616	.75	1

Table 8 - Metrics for Simulation Group 2

Simulation	$T_{extended}$	$T_{modified}$	$\bar{\mu}_{extended}$	$\bar{\mu}_{modified}$
2.1	4	119	1.158	8.792
2.2	4	127	1.211	7.015
2.3	4	129	1.198	7.816
2.4	4	132	1.215	6.723
2.5	4	132	1.214	6.355
2.6	4	135	1.215	5.358

Table 9 - Summary Statistics for Simulation Group 2

In simulations 2.1 and 2.2, I seek to see the effect of varying the number of features per object on a single group. The only difference between simulation 2.1 and 2.2 is that simulation 2.2 has, on average, more shared features per word within the first group. The graphs of the trials for the extended network for both simulations are presented below (Figures 24 and 25):

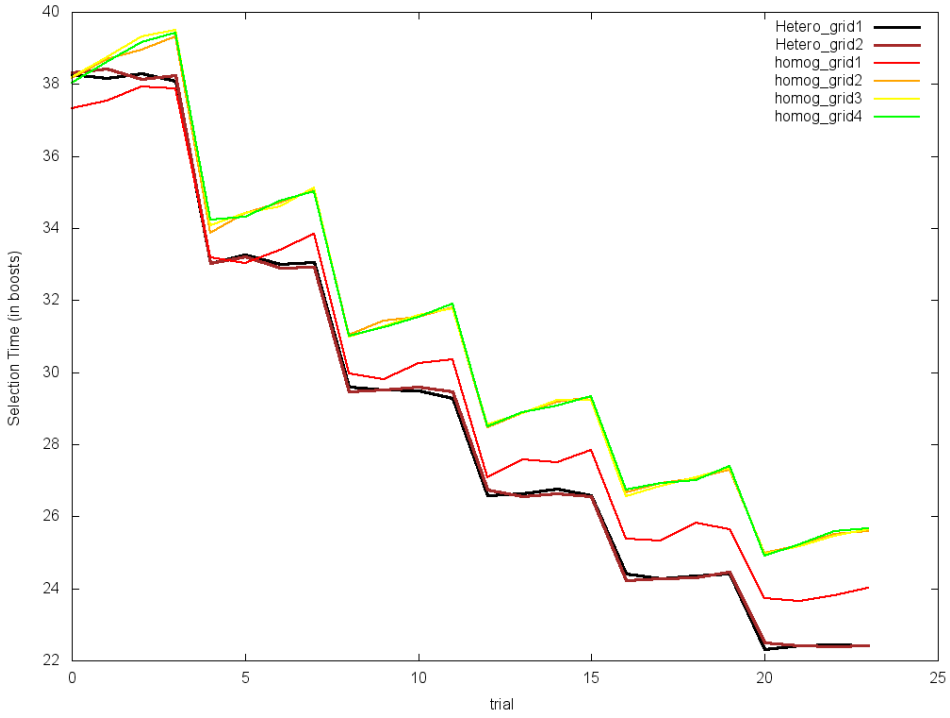

Figure 24 - Simulation 2.1, extended network

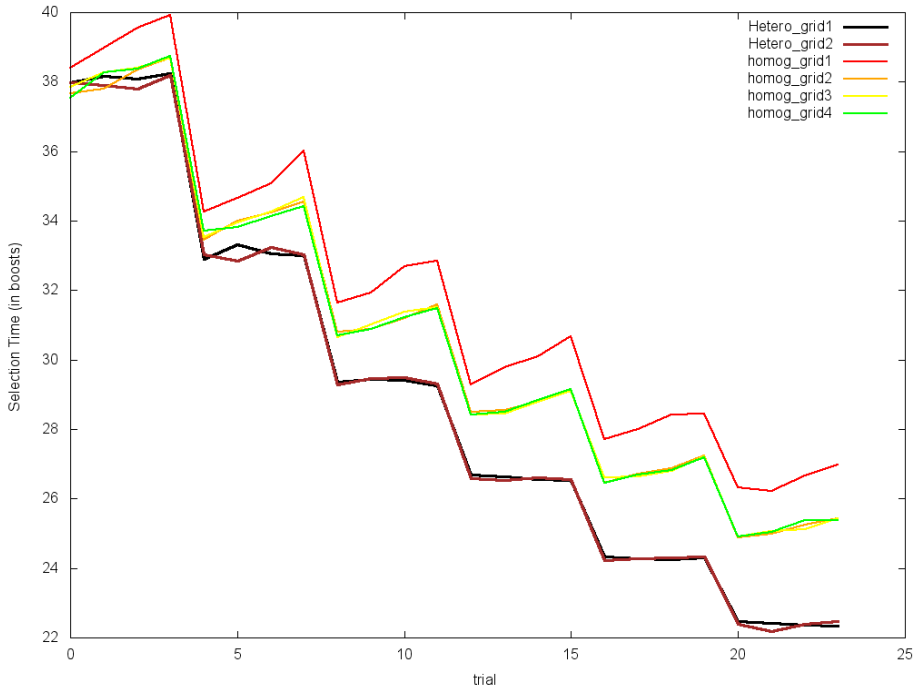

Figure 25 - Simulation 2.2, extended network

From these figures, we can clearly see the predictive power of the value of D_A. Recall that in the first graph, D_A of group 1 (the red line) is .639, while in the second it is .541. Furthermore, the D_A of the other three groups in both graphs is .581 – and the D_A of the heterogeneous group is .75. If we were to order these groups according to their homogeneity based solely on their D_A values, we would predict an ordering of:

$$G_H < G_1 < G_{2,3,4}$$

for the first simulation and:

$$G_H < G_{2,3,4} < G_1$$

for the second. The ordering of the relative boost counts of each group reflects this relationship.

An important thing to note here is the position of the heterogeneous groups with respect to the homogeneous groups. Because the heterogeneous groups are composed of elements of the homogeneous groups, we expect the heterogeneous groups' boost counts at trial 0 to be positioned at approximately the average of the of the homogeneous groups' boost counts at trial 0, assuming the boost counts within each homogeneous groups have *low variance*. If we see the heterogeneous groups beginning at a boost count far away from expectation, we can conclude that this positioning is due to high variance in the boost counts in the homogeneous groups. We will see this effect in the modified network. I describe outputs with outlying boost counts as *antagonistic outputs*. These will be described in greater detail shortly.

Examining the same graphs as output by the modified network shows the same effect as above (Figures 26 and 27):

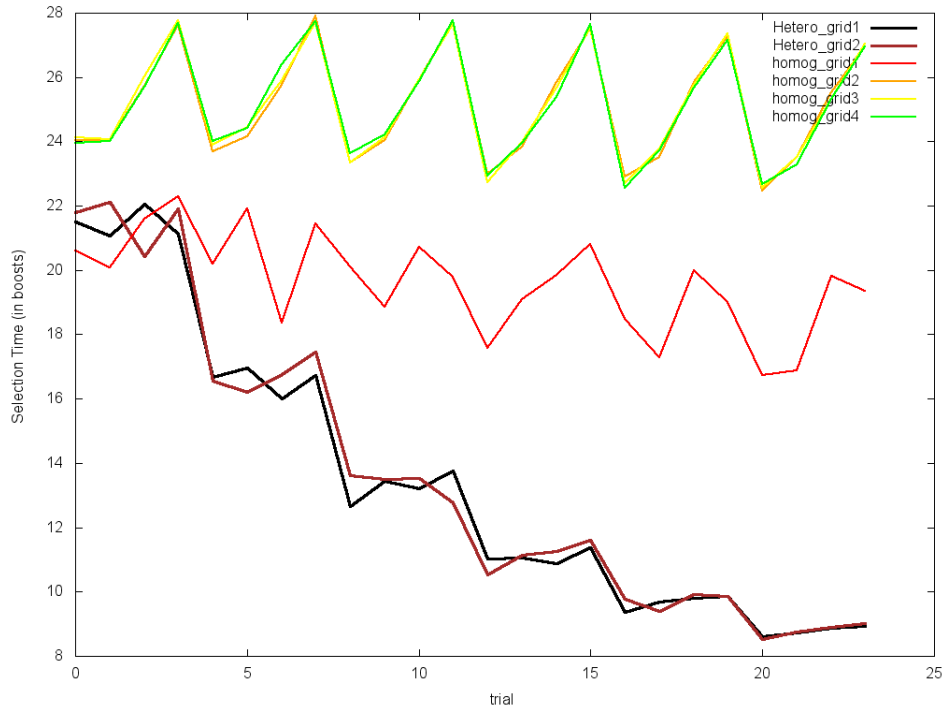

Figure 26 - Simulation 2.1, modified network

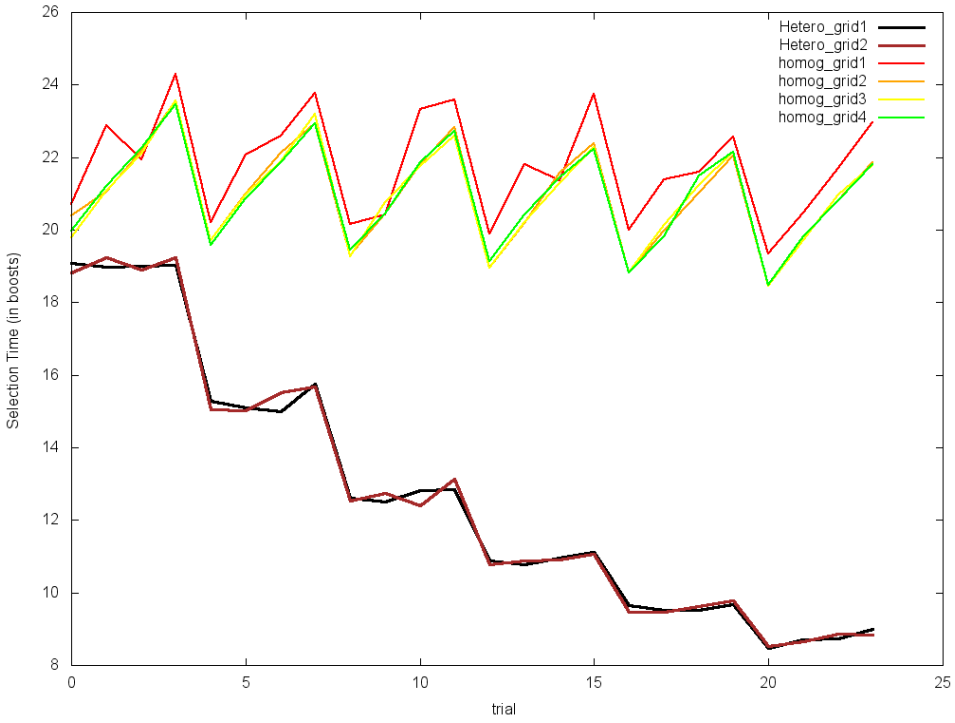

Figure 27 - Simulation 2.2, modified network

While we see the same behavior as predicted, we also see some interesting behavior typical of the modified network (see Section 4.3.3). What we see here is an output indicative of a network that has not immediately converged to the correct solution. Indeed, as evidence for this, the training graph is shown below:

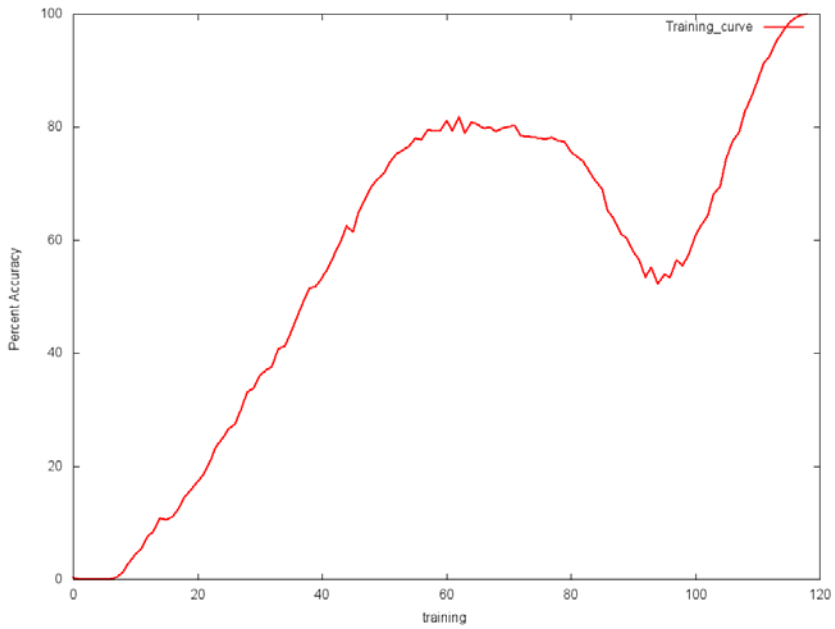

Figure 28 - Simulation 2.1, Training Curve, modified network

Note that this curve is not monotonically increasing, indicating that the network is oscillating through the error space in a way I wish to avoid. However, once the network converges to an error-minimizing location, we do see an output that exhibits all the behavior we expect (see Figure 27).

In order to explain the shape of this particular output (Figure 28), it is important to understand what is preventing the immediate convergence of the network. Generally, convergence is prevented by one or two outputs across multiple groups, which antagonize each other's connectivity with their respective outputs – the aforementioned *antagonistic outputs*. Because of this, while these two or three outputs slowly approach a correct

solution, other outputs become *overtrained*. Thus, when we finally see the net testing output, the homogeneous groups exhibit little to no repetition priming, as most of their outputs' connections are already at optimal weights. Meanwhile, the heterogeneous group, which contains the antagonistic outputs, improves rapidly as we expect. Thus we see much higher discrepancies between the heterogeneous groups and the homogeneous groups in cases where an error minimizing location is not immediately approached. The relative ordering of the groups is correctly maintained, however.

This behavior suggests a different method for calculating the metric chosen to represent the relative amount of cumulative semantic interference observed – to average only the positively sloped regions of all of the curves and take their ratio. This would be reasonable, but this metric is generally very difficult to calculate for real data, as the regions are very short, and would thus require the repetition of the experiment many hundreds of times to generate a reasonable estimate for each slope.

The next two simulations operate as a sort of converse to the first two – I now fix the number of features (and thus the D_A) of group 1 and vary the other three groups. Simulations 5 and 6 vary all four groups. For the extended network, the behavior is exactly as described above. Simulation 2.6's output is shown below (Figure 29). From the values of D_A, we expected the following ordering:

$$G_H < G_4 < G_1 < G_3 < G_2$$

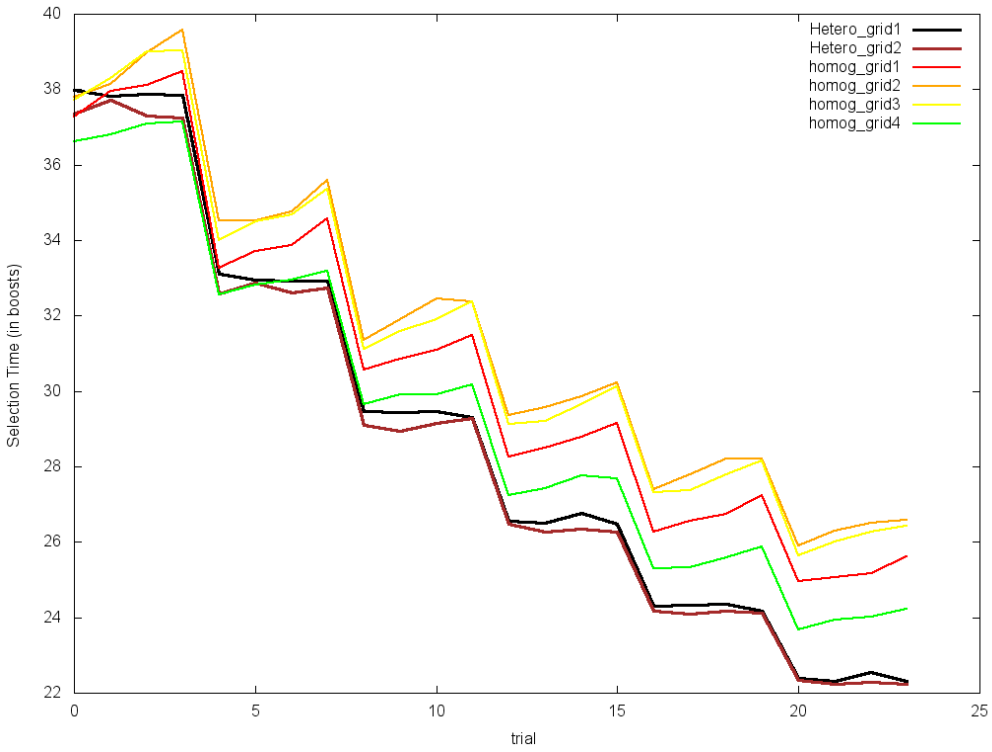

Figure 29 - Simulation 2.6, extended network

This is precisely the ordering we see. However, strangely, for these four runs, the modified network produces a nearly *opposite* ordering, shown below for Simulation 2.6 (Figure 30):

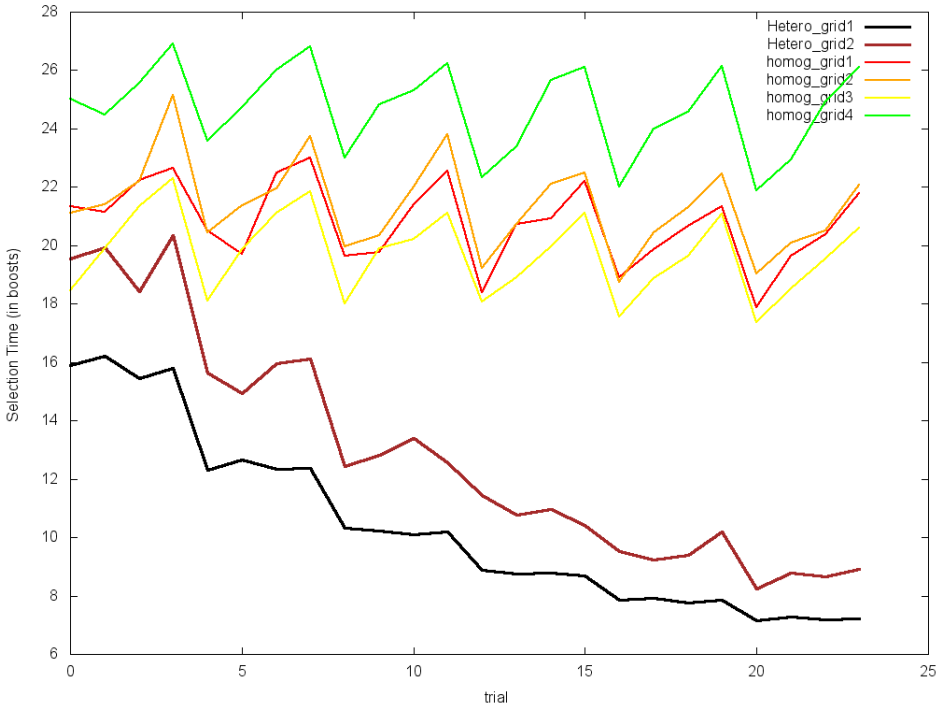

Figure 30 - Simulation 2.6, modified network

This is the first example of a situation wherein the "antagonistic inputs" negatively affect the results of the modified network's output. Unfortunately, without modifying the learning rule, cases like these will sometimes occur.

Consider for a moment the operation mode of the modified network. When we introduce an input, that input is immediately transformed via the *activateSecondary* function – the input vector is scaled and rotated towards semantically related features' dimensions. Remember that D_A is essentially a measure of distance. If this average distance is low, we expect high boost counts, as the output vectors are spatially close and

are thus difficult to distinguish from another. Unfortunately D_A does not take into account the scaling and rotation caused by *activateSecondary*. However, we can roughly predict the location of the resultant transformed vector by noting the result from Section 4.3.2. We know that the secondary activation strengths are proportional to the square of the connection weights in the network – and since these connection weights are between 0 and 1, this will tend to make lower weights far less significant than higher weights. Thus, if I design a metric to look at the average of the pairwise maximum shared feature between words in a group, we might be able to correct our prediction. Formalizing this, I take the Chebyshev distance in the formula for D_W instead of the Euclidean distance – this gives us:

$$D_{W_{modified}}(w_a, w_b) = \frac{\lim_{n\to\infty} \sqrt[n]{\sum_k (f_{k_a} - f_{k_b})^n}}{\sqrt{2}}$$

and use this modified formula to calculate D_A for the groups in the modified network. For Simulation 2.6, this gives us:

Simulation	Group 1 Modified D_A	Group 2 Modified D_A	Group 3 Modified D_A	Group 4 Modified D_A	Heterogeneous Group Modified D_A	Modified $\overline{D_G}$
2.6	.269	.286	.282	.227	.310	.358

Table 10 - Modified D_A for Simulation 2.6

which predicts the ordering:

$$G_H < G_2 < G_3 < G_1 < G_4$$

We can average these with our previous predictions after normalization:

$$D_{A_{adjusted}} = \frac{\frac{D_A}{\overline{D_G}} + \frac{D_{A_{modified}}}{\overline{D_{G_{modified}}}}}{2}$$

which (in this case) gives us the same ordering. Unfortunately, this still does not give us the *correct* ordering (it underrepresents G_2) – but it does correctly place G_4 as the most homogeneous group. We will see later that using this adjusted D_A for the modified graph tends to have more predictive power than the naïve approach over the homogeneous groups. The heterogeneous groups are still governed by the original D_A, as they have few shared features.

5.2.4: Simulation Group 3

In Simulation Group 3, I complete the transition from synthetic, engineered data to naturally derived feature norms. All simulations conducted in this Group consist solely of data directly extracted from the McRae feature norm dataset.

For this group, I extract 4 sets of 4 semantically related concepts from the McRae feature norms. These sets serve as our homogeneous groups. As before, a heterogeneous group of 4 objects is constructed via the homogeneous groups by including one object from each. I repeat this process 4 times to produce 4 separate heterogeneous groups which share no objects among them. This simulation configuration is then tested over 4 different data sets. To do this, I embed the 16 objects' features in larger feature spaces. I do this in order to measure the effect of extraneous knowledge on the performance of the networks. For the extended network, I expect these additional features to have little to no effect, as they are never directly excited by any input during the testing phases. For the modified network, however, I expect to see some change in the output, as I expect these extraneous inputs to be secondarily excited by semantically related primary inputs during the testing phases.

Four different embeddings were chosen; the first of these is a simple control group, which contains no features other than the features belonging to the 16 objects of the homogeneous groups. The next embedding contains features from a set of additional words which share many features with the 16 objects in the homogeneous groups. For example, this embedding includes the set of features describing words such as "chickadee" and "crow", which are highly semantically related to the bird group. The third embedding contains a set of features belonging to words that are semantically *unrelated* to the base set of 16 objects. Finally, the fourth embedding includes the sets of features corresponding to both the second and third embeddings.

In all four cases, the network is trained on the additional words, but only tested on the base 16 words. A summary of the four simulations is presented below (Tables 11-13):

"Birds" D_A	"Tools" D_A	"Instruments" D_A	"Clothing" D_A	$\overline{D_G}$
.474	.629	.521	.650	.986
Heterogeneous Group 1 D_A	**Heterogeneous Group 2 D_A**	**Heterogeneous Group 3 D_A**	**Heterogeneous Group 4 D_A**	
.737	.744	.738	.745	

Table 11 - Metrics for Simulation Group 3

Simulation	Includes Extraneous Homogeneous Word Set	Includes Extraneous Heterogeneous Word Set	Total Feature Count	Total Word Count
3.1	No	No	133	16
3.2	Yes	No	180	24
3.3	No	Yes	231	28
3.4	Yes	Yes	274	36

Table 12 - Simulation Descriptions for Simulation Group 3

Simulation	$T_{extended}$	$T_{modified}$	$\bar{\mu}_{extended}$	$\bar{\mu}_{modified}$
3.1	7	907	1.315	8.025
3.2	7	737	1.276	2.971
3.3	6	851	1.292	6.856
3.4	7	712	1.267	2.75

Table 13 - Summary Statistics for Simulation Group 3

Simulation 3.1 serves two purposes: first, it serves as a witness to the validity of the claim that both the extended and the modified networks successfully model semantic interference using real-world data. Second, it serves as a baseline with which to compare the other three simulations in Group 3. I present its boost count outputs across all cycles below (Figure 31):

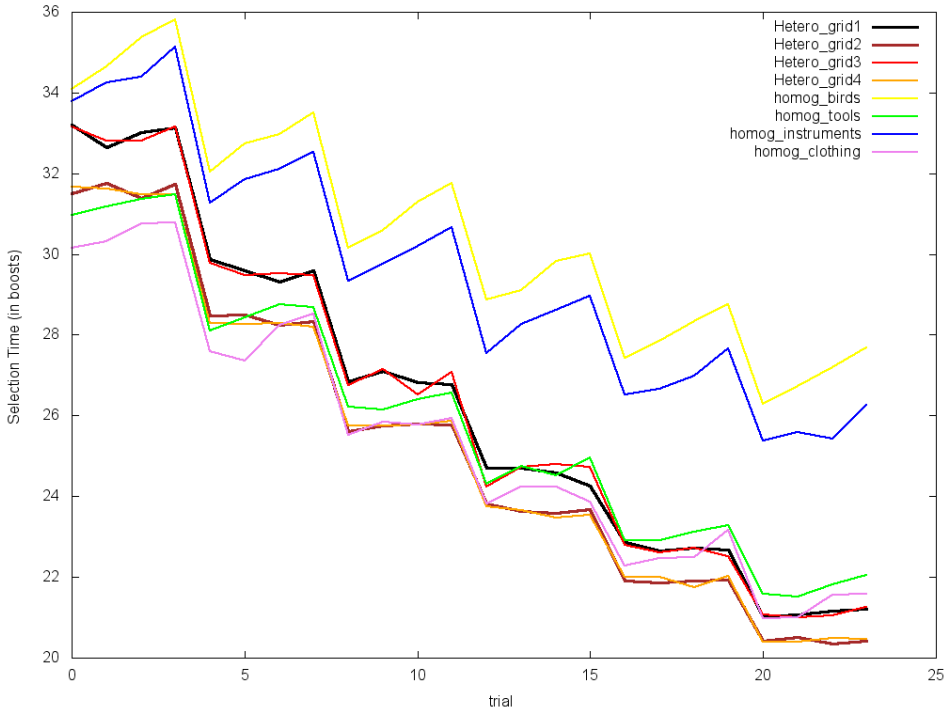

Figure 31 - Simulation 3.1, extended network

First, we indeed see evidence for both repetition priming and semantic interference from the shape of the output of this graph. Furthermore, we see the heterogeneous groups'

boost count at trial 0 sitting close to the average of the homogeneous groups' boost counts for this trial, indicating low variance in the boost counts over all groups. This in turn indicates that all outputs are trained equally well. Furthermore, the training period of this 133 feature count dataset, the largest yet used in the simulations, was only 7 training epochs, showing good scaling performance (consider the data from Simulation Group 1 for comparison). Finally, the ordering of the four groups' boost counts is correctly predicted by the ordering of the D_A metric of each group.

The discrepancies between the modified network's results and the extended networks results are similar to the differences between the results of Simulation 2.6 (Figure 32):

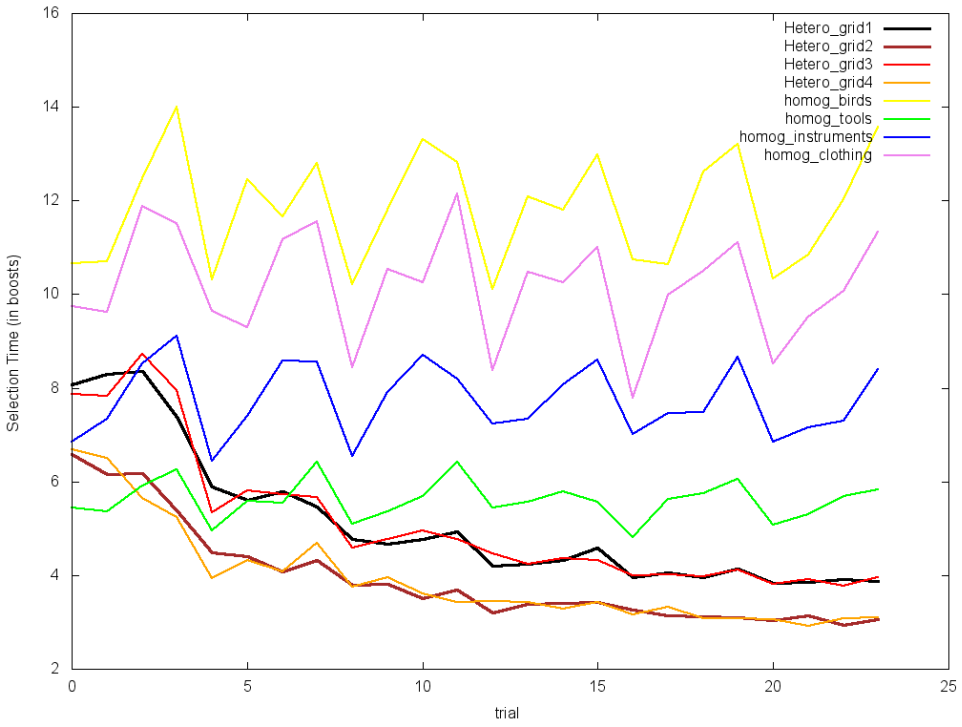

Figure 32 - Simulation 3.1, modified network

Here we see the "clothing" homogeneous group appearing much higher than expected, and the "instruments" group appearing much lower. Let us calculate the modified D_A from Section 5.2.3 over this data (Tables 14 and 15):

"Birds" Modified D_A	"Tools" Modified D_A	"Instruments" ModifiedD_A	"Clothing" Modified D_A	Modified $\overline{D_G}$
.089	.247	.097	.046	.380
Heterogeneous Group 1 Modified D_A	**Heterogeneous Group 2 Modified D_A**	**Heterogeneous Group 3 Modified D_A**	**Heterogeneous Group 4 Modified D_A**	
.297	.307	.272	.231	

Table 14 - Modified D_A for Simulation Group 3

"Birds" Adjusted D_A	"Tools" Adjusted D_A	"Instruments" AdjustedD_A	"Clothing" Adjusted D_A
.357	.644	.391	.390
Heterogeneous Group 1 Adjusted D_A	**Heterogeneous Group 2 Adjusted D_A**	**Heterogeneous Group 3 Adjusted D_A**	**Heterogeneous Group 4 Adjusted D_A**
.765	.781	.732	.683

Table 15 - Adjusted D_A for Simulation Group 3

Predicting the ordering over this data gives us:

$$G_{H_2} < G_{H_1} < G_{H_3} < G_{H_4} < G_{tools} < G_{instruments} < G_{clothes} < G_{birds}$$

However, since the heterogeneous groups share very few features, it makes more sense to use the original D_A instead of the adjusted D_A for these groups. Doing so gives us the ordering:

$$G_{H_4} < G_{H_2} < G_{H_3} < G_{H_1} < G_{tools} < G_{instruments} < G_{clothes} < G_{birds}$$

which is the correct ordering.

Unfortunately we will see that this adjusted metric completely loses its predictive power for the simulations that contain extraneous concepts, as it has no mechanism for taking their features into account.

I present the learning curve of the modified network on Simulation 3.1 below for comparison with later simulations (Figure 33):

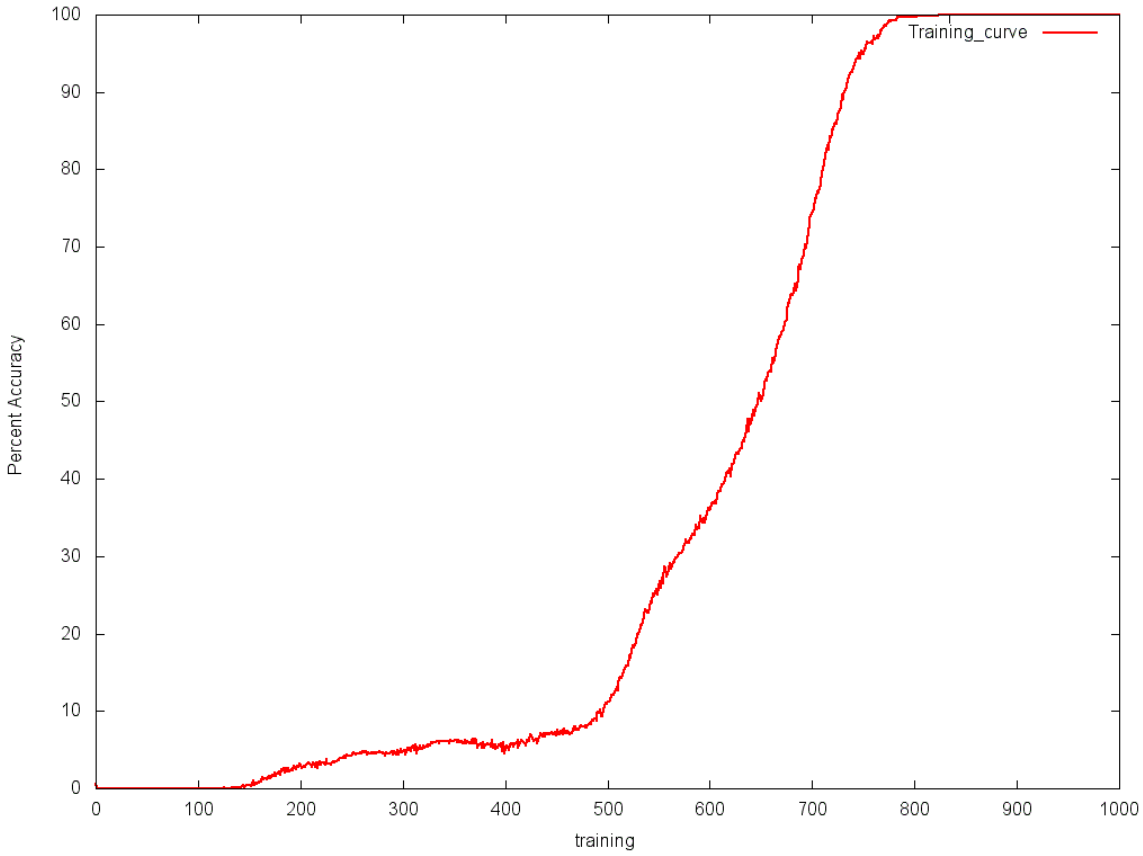

Figure 33 - Simulation 3.1 training curve, modified network

In Simulation 3.2, I introduce an additional 2 words per group that are learned by the network but not tested for. I expect very little change in output for the extended network, but expect significant differences in the output for the modified network, as it should involve these inactive words in all of its decisions. The extended network's output for Simulation 3.2 is shown below (Figure 34):

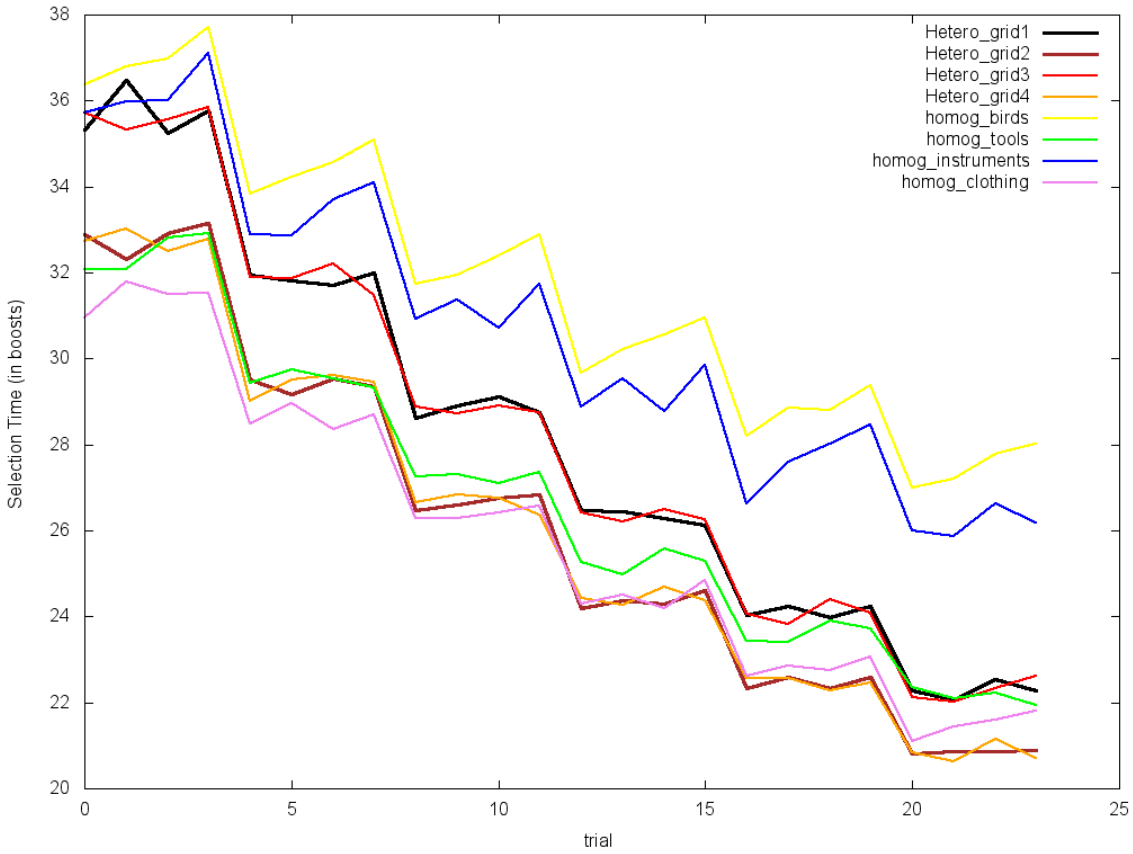

Figure 34 - Simulation 3.2, extended network

As expected, this output closely matches the output from Simulation 3.1. The calculated values for $\bar{\mu}_{extended}$ also match quite closely. We see an overall upwards shift in the graph compared to Simulation 3.1 (Figure 31) – this is due to the additional extraneous output units contributing to the average competitor weight for each boost calculation, increasing boost counts unilaterally. All other variations are due to the noise parameter θ.

Now I examine the output for the modified network over the same simulation (Figure 35):

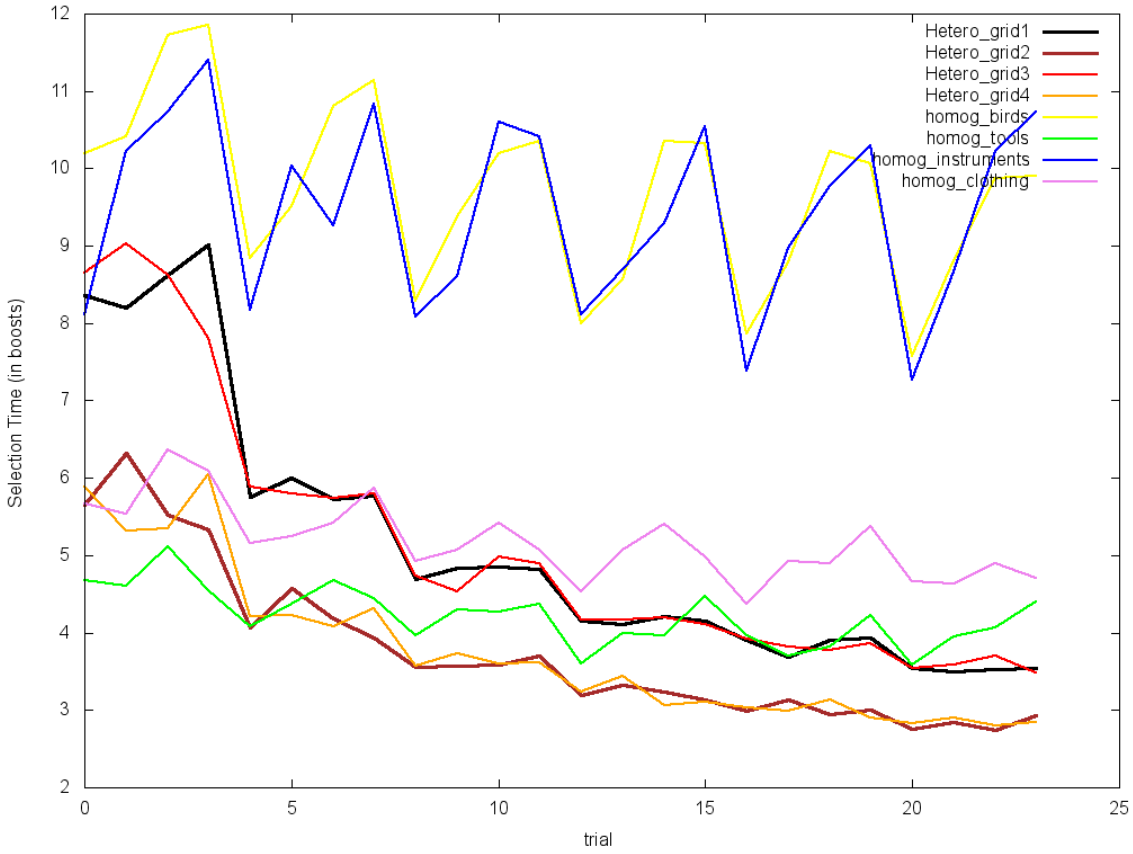

Figure 35 - Simulation 3.2, modified network

We can see many clear differences between this graph and the graph of Simulation 3.1 (Figure 31), most notably the inversion of the "instruments" and "clothing" groups. This indicates that the network is indeed taking into account inputs that are not directly excited as we expect. Unfortunately, it is very difficult to predict the network's behavior by studying the structure of the added extraneous features.

Two important features of this simulation should be noted: even though I *added* input and output units to this network, the overall training period has gotten smaller.

Additionally, the $\bar{\mu}_{modified}$ metric has decreased as well, mostly due to the decrease in magnitude of the slope of the heterogeneous conditions. This value for $\bar{\mu}_{modified}$ is more reasonable than the previous simulation's, as the homogeneous conditions' slopes in this simulation can be more accurately estimated with a linear regression. Thus we see that the addition of homogeneous units to the modified network can help alleviate the problem of "antagonistic outputs" by coercing them into converging faster, reducing the amount of overtraining via reducing the overall training period.

In Simulation 3.3, I add extraneous heterogeneous words to the base simulation. Because I am using a real feature norm dataset, these heterogeneous words are not fully heterogeneous to the four extant groups. The group I chose consisted of 4 foods, and so will be referred to as G_{foods}.I show the cross differences below (Table 16):

$D_G(G_{birds}, G_{foods})$	$D_G(G_{tools}, G_{foods})$	$D_G(G_{instruments}, G_{foods})$	$D_G(G_{clothing}, G_{foods})$
.990	.997	.993	.998

Table 16 - Simulation Group 3 Extraneous Heterogeneous Group Cross Differences

Examining this table, we can roughly predict changes tending upwards in the positions of both the "bird" and "instrument" groups for the modified network. I still expect little to no change in the extended network.

First, I present the extended network's output (Figure 36):

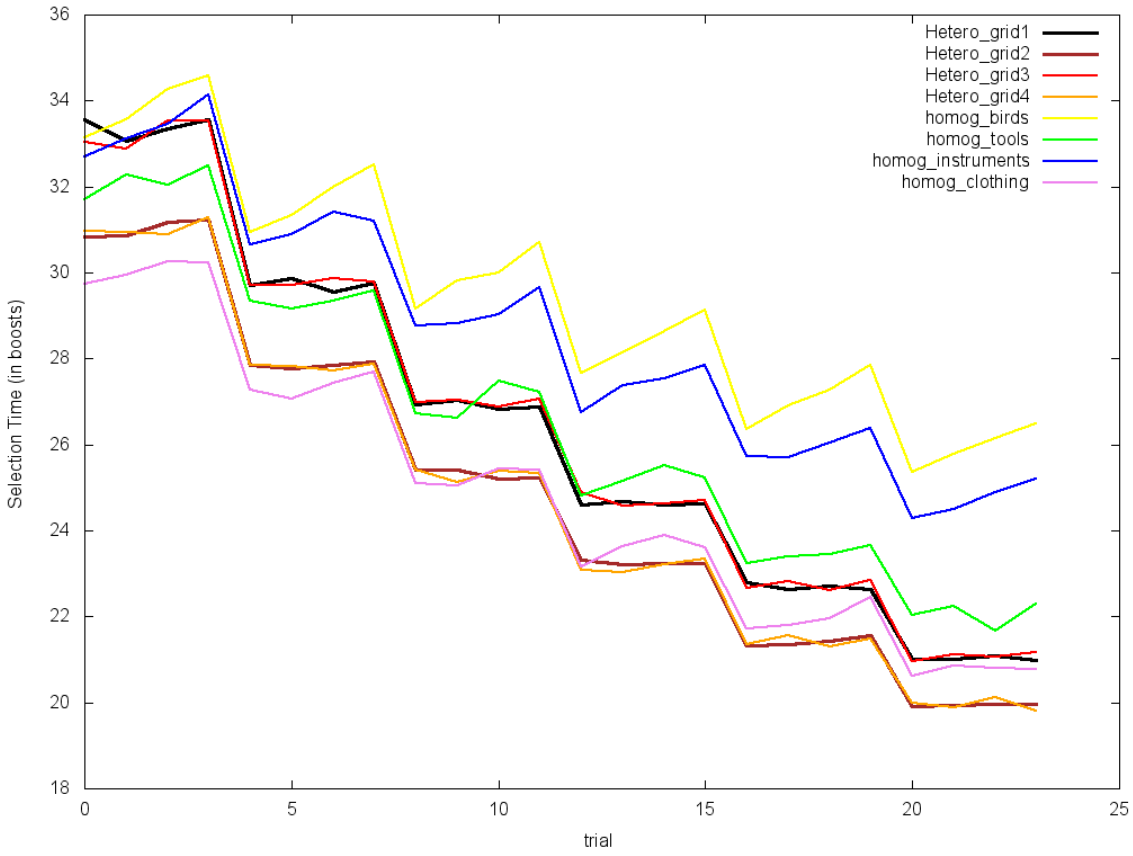

Figure 36 - Simulation 3.3, extended network

We see here an overall reduction in boost count unilaterally, but no relative changes in the simulation, as expected. The reduction in boost count is due to the extraneous heterogeneous features *decreasing* the competitors' average activation during the boosting calculations.

Now, I examine the modified network's output (Figure 37):

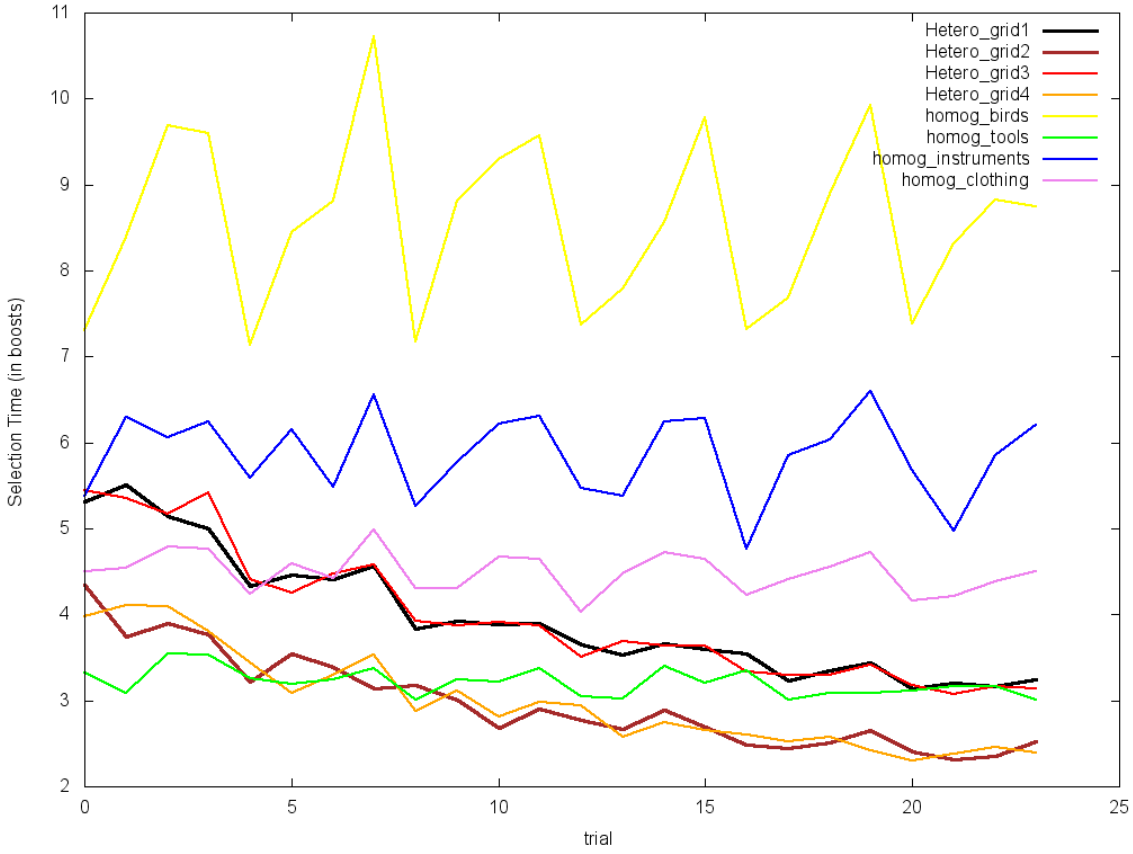

Figure 37 - Simulation 3.3, modified network

For reference, compare the relative average boost counts of groups "instruments" and "birds" here to the output in Figure 32. We see the predicted increase in boost counts for each, due to their lower cross difference with the added extraneous heterogeneous features. We also see a slight decrease in $\bar{\mu}_{modified}$, in spite of the fact that was have a *longer* training period. This is because the small amount of shared features between the extraneous heterogeneous group and two of the four homogeneous groups is actually helping the network as before, by reducing the amount of overtraining. However, the

overall increase in heterogeneity of the network necessitates a longer training period. Therefore, we see an overall increase in *T* with a net *decrease* in overtraining.

Let us also examine the learning curve for the modified network for Simulation 3.3:

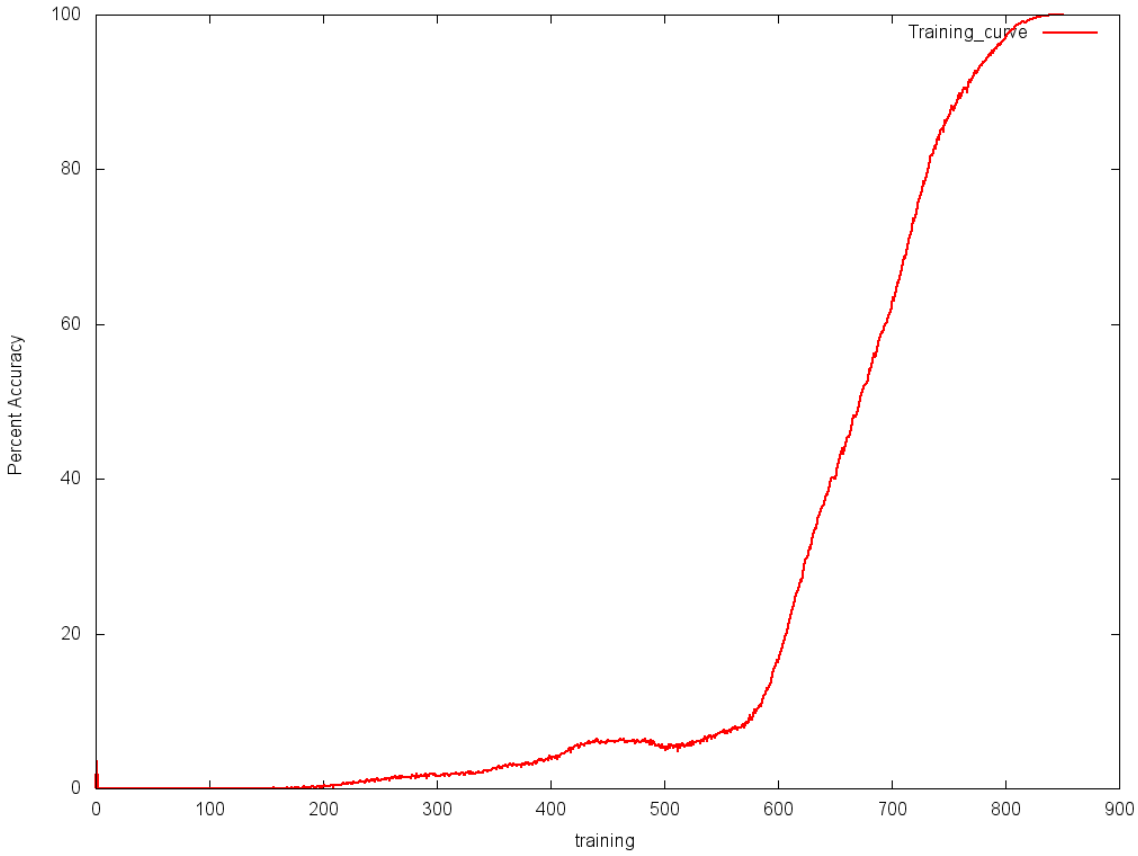

Figure 38 - Simulation 3.3 training curve, modified network

Note the similarity between this curve and the curve of Figure 33. Indeed, this curve can be seen as a simple horizontal scaling of the earlier learning curve; if the added words were entirely heterogeneous, this curve would be an exact horizontal scaling of the original, corresponding to a simple increase in *T* with no behavioral changes to the network. However, because some of the homogeneous groups share features with the

extraneous heterogeneous words, we see a slight smoothing of this curve (most noticeable during the final ascent to 100% accuracy). This smoothing corresponds to a reduction in overtraining, as it reflects a reduction of the amount of epochs wasted moving away from the local error minimum.

Finally, Simulation 3.4 includes both extraneous groups, resulting in a 36 word network. In the extended network, we see little change, as before. A shift upwards from the extraneous homogeneous features is counteracted by a shift downwards from the heterogeneous conditions, with a negligible net effect. The modified network benefits from the improvement in performance offered by both sets of extraneous features, and therefore produces its lowest $\bar{\mu}_{modified}$ with the shortest $\boldsymbol{T}$. Shown below are both networks' outputs (Figures 39 and 40):

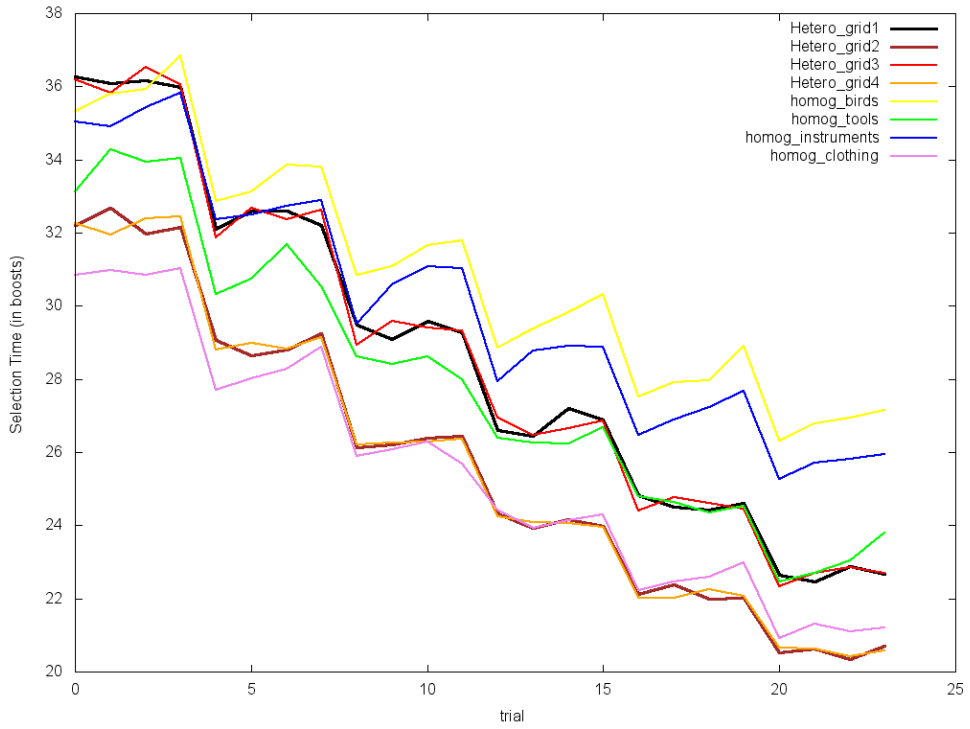

Figure 39 - Simulation 3.4, extended network

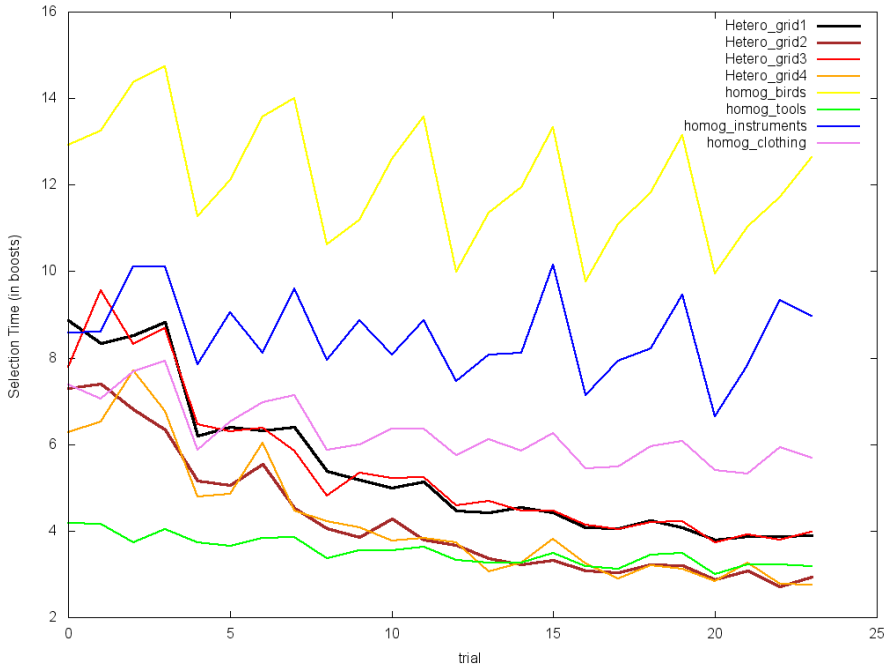

Figure 40 - Simulation 3.4, modified network

Also interesting is the modified network's training curve, which is the smoothest (and shortest) yet produced (Figure 41):

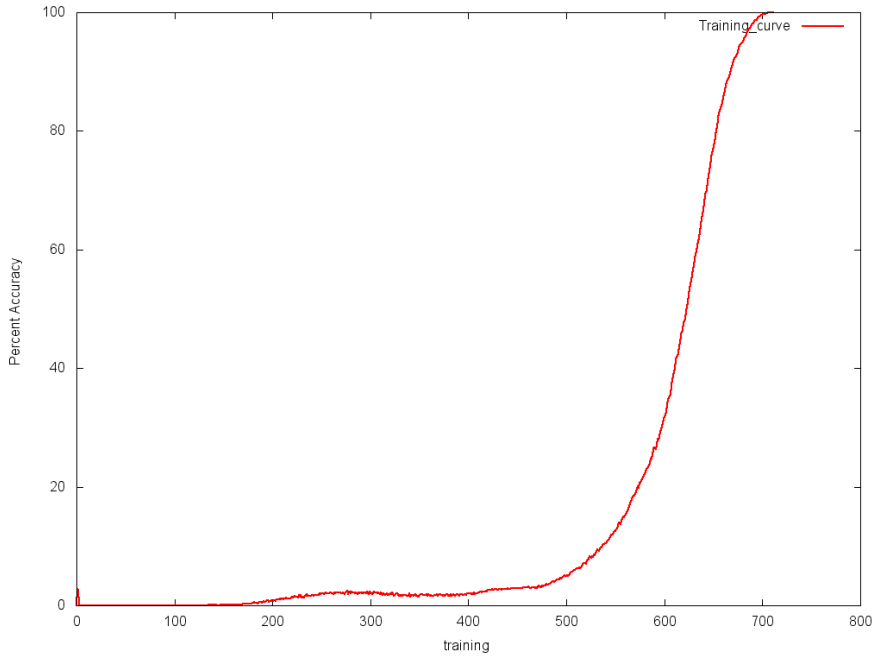

Figure 41 - Simulation 3.4 training curve, modified network

5.2.5: Noise Tolerance

An important property of any computational model that attempts to operate on real-world data is its ability to function properly in spite of noisy inputs. With this in mind, I devised a test for evaluating both networks' performance with increasing amounts of noise. I chose a simple simulation from Simulation Group 1 (5 features per object, 3 shared features per group, no features shared across groups, 4 total groups) as a neutral testing platform and simulated noisy inputs by increasing the internal noisiness of the network itself – this noise directly affects the connection weight updates and all network units' output values. This particular simulation was chosen as the modified network had a monotonically increasing learning curve on it (i.e. immediately converged to an error-minimizing set of connection weights) and because it lay in the center of the region of test coverage. I then graphed the maximum accuracy achievable in the testing phase against the noise parameter θ, which acts as the standard deviation of a normal distribution (centered at 0) from which I sample noise vectors that are added to connection weights and unit outputs throughout the network. When the simulation could not reach 100% accuracy during the testing phase, the value of its asymptote at 1000 epochs was taken as its maximum.

It should also be noted that although the networks can achieve 100% accuracy for many of the noise parameter values given below, their outputs are generally illegible for higher noise values, i.e., the network correctly classifies the output but does it in what is essentially a random time (boost count) and so does not necessarily produce an output that is easily interpretable as a demonstration of semantic interference.

The results for both networks are shown below:

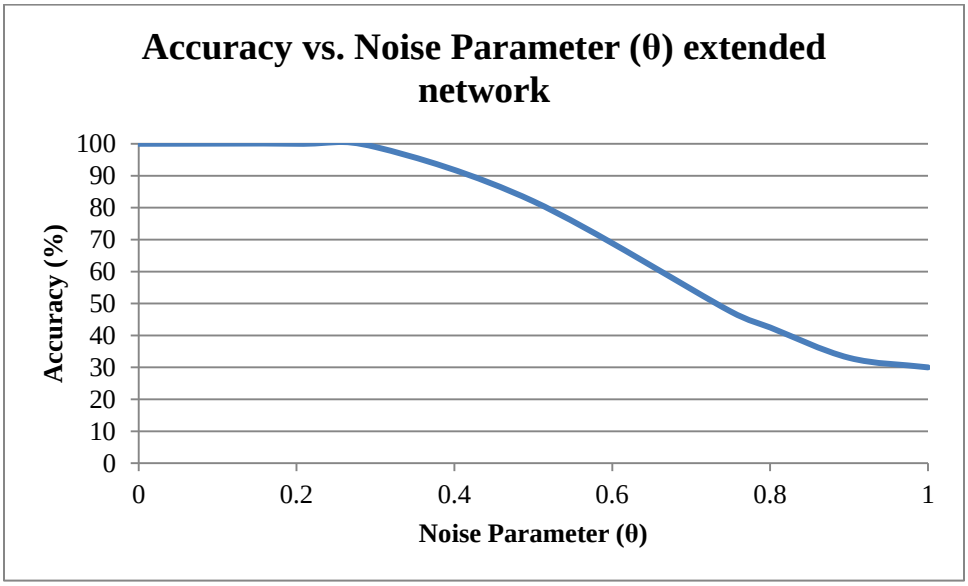

Figure 42 - Accuracy vs. Noise, extended network

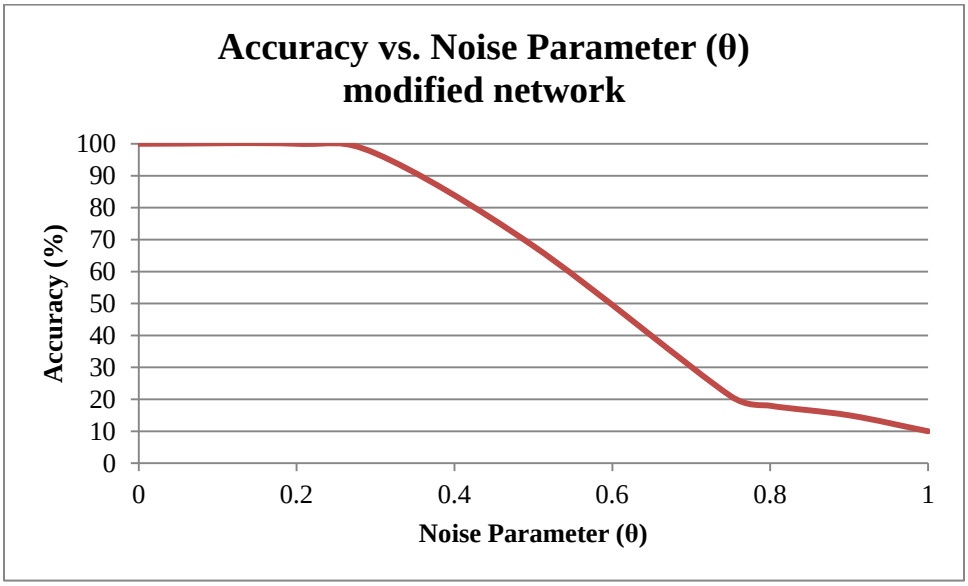

Figure 43 - Accuracy vs. Noise, modified network

Both networks perform well up until the 0.2 standard deviation mark, where performance starts to degrade (Figures 42 and 43). The performance degradation occurs faster in the modified network, which is more sensitive to noise overall. However, both networks are

relatively robust to noise, maintaining near 100% accuracy even in the face of errors of $\sim \pm .2$ excitation weight.

5.2.6: Longevity Testing

One final property of the networks that I explored sought to understand how "long" the learned effects of semantic interference lasted. Because neural networks do not naturally change or degrade over time, I needed to emulate the passage of time via other means. The following steps were performed in these simulations in order to approximate the effects of time passing:

1. Train the network to 100% accuracy.
2. Test the network on all homogeneous conditions consecutively, in random order. Do not reset the network during or after testing.
3. Train the network a variable number of epochs again. The number of epochs here corresponds to the time that has passed.
4. Re-test a copy of the resultant network on each homogeneous and heterogeneous condition and collate the results

Thus, I emulate the passage of time via a variable number of complete presentations of the word set in a random order. By presenting the words in a random order, and not in close proximity to other semantically related words, I emulate the natural course of retrieval of such words over time. I then retest to see how much semantic interference the network produces later. If the amount of interference increases, I can conclude that the interference effects produced via the learning mechanism in the

testing phase have remained in memory in spite of the interstitial period of random word use.

I simulated the results of the above procedure on both networks over Simulation 3.4 – the most complex dataset. I then graphed the $\bar{\mu}$ metric against the number of epochs used in step 3 of the above procedure. The results are reproduced below (Figures 44 and 45):

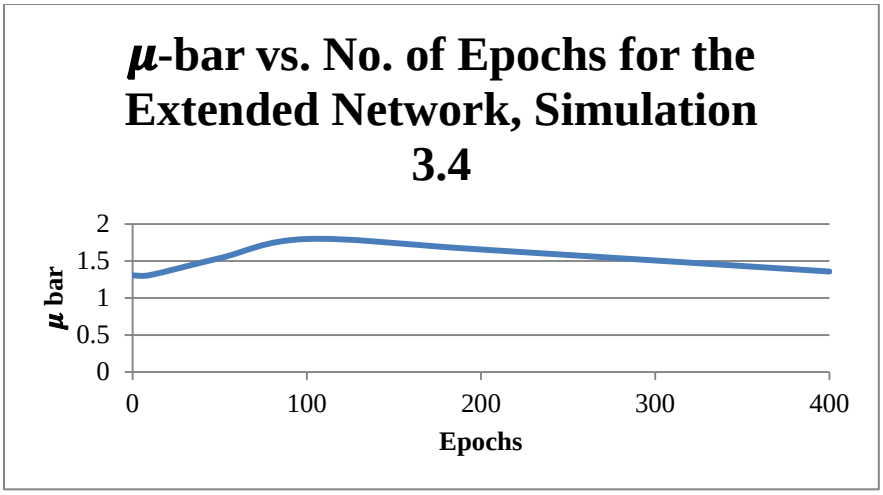

Figure 44 - Longevity Testing, Extended Network

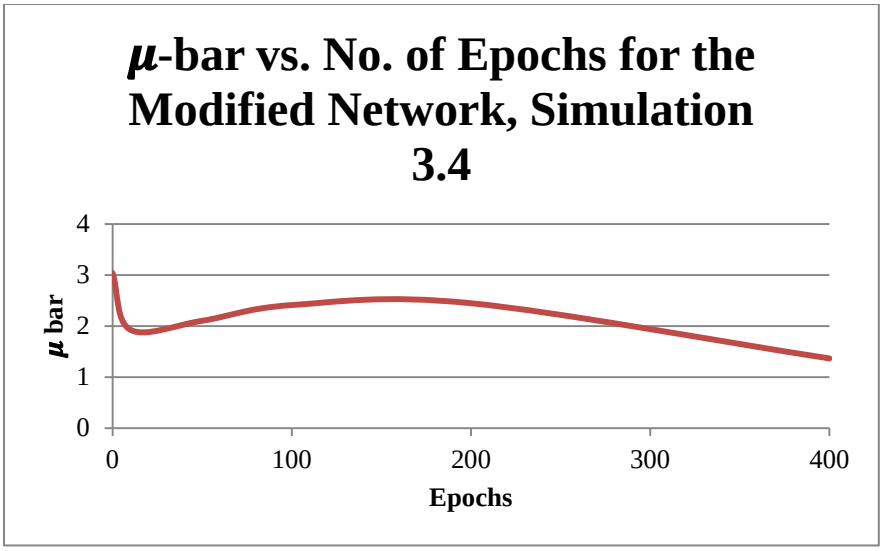

Figure 45 - Longevity Testing, Modified Network

Examining the figures, both networks show an initial period of *increasing* interference. This can be attributed to the experimental method; because the interstitial period consists almost entirely of words that are also members of the testing set, both networks enjoy a period of further optimization beyond the 100% accuracy mark that will continue to reduce overall boosting rates due to refinements in the selectiveness of the connection weights. After this period is over, however, we see the expected reduction in interference effects in both graphs. It seems that the modified network maintains the state of maximum interference for longer than the extended network. However, they are difficult to compare directly; the modified network requires an initial training period of approximately 700 epochs, while the extended network only requires approximately 7 epochs. The initial spike in the modified network graph is due to the "antagonistic outputs" as explained earlier – once the network has converged to 100% accuracy, we see that these outputs quickly revert to their normal behavior, thus dropping the $\bar{\boldsymbol{\mu}}$ measures to more reasonable levels.

The above graphs are deceiving, however, because of the choice of metric. While $\bar{\boldsymbol{\mu}}$ is useful for measuring the difference in interference across equally trained networks, it is not necessarily a good measure of semantic interference when comparing networks with different training periods. I select a number of the points above for presentation below in order to evaluate the effects of semantic interference manually. I present the data for (Epochs = 0), (Epochs = 10), (Epochs = 50), and (Epochs = 100) for the extended network (Figures 46, 47, 48, and 49):

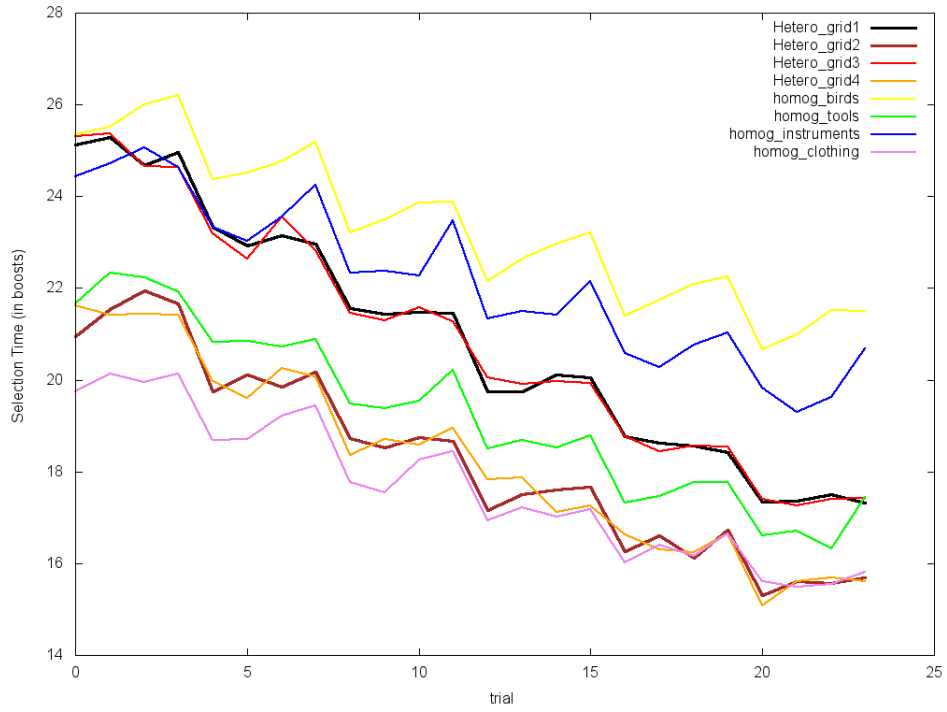

Figure 46 - Longevity Test, Epochs = 0, extended network

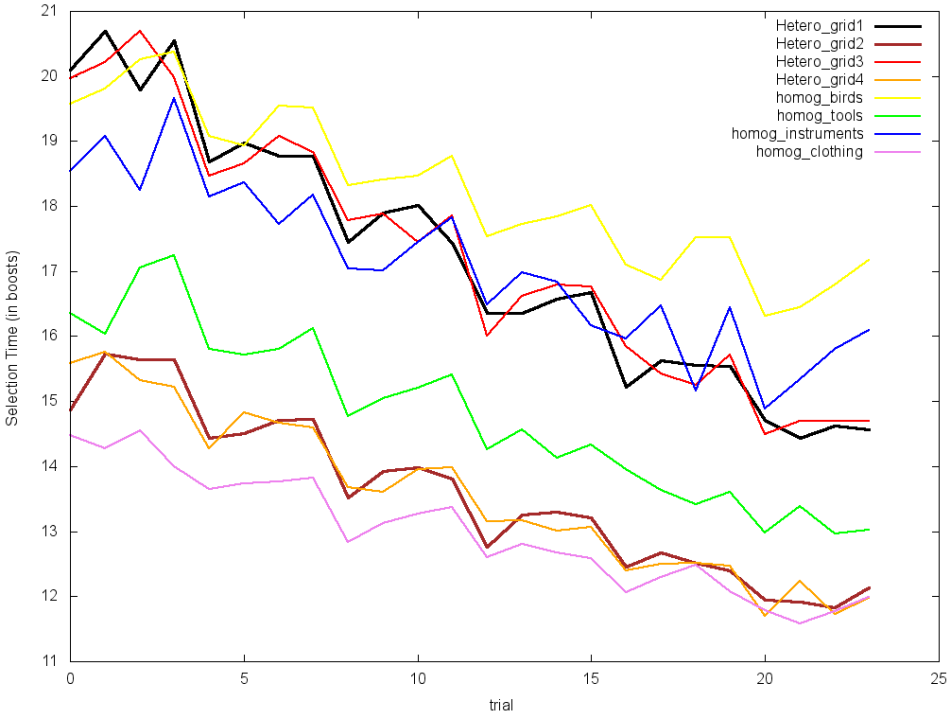

Figure 47 - Longevity Test, Epochs = 10, extended network

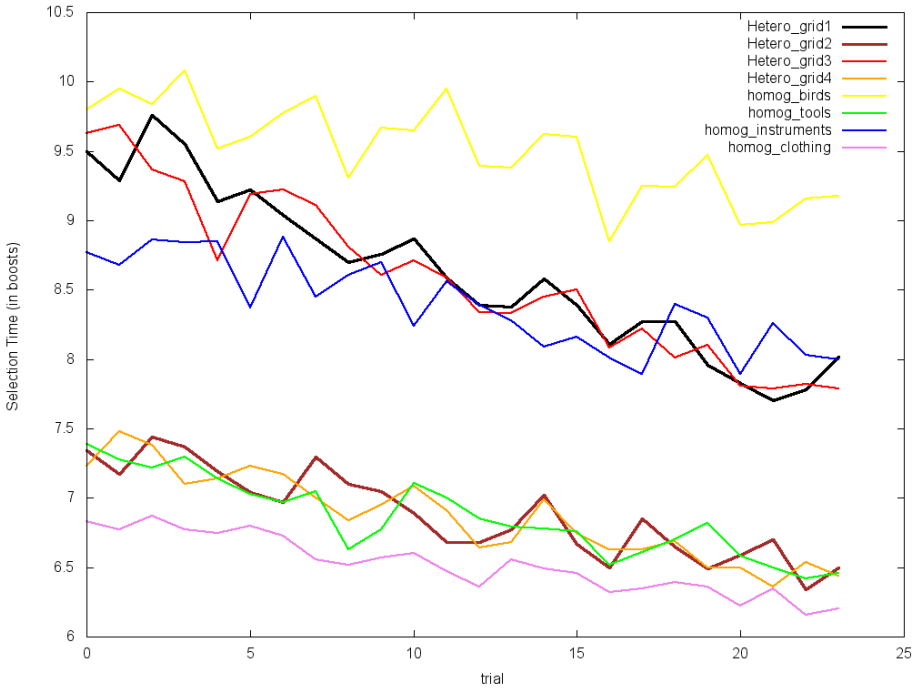

Figure 48 - Longevity Test, Epochs = 50, extended network

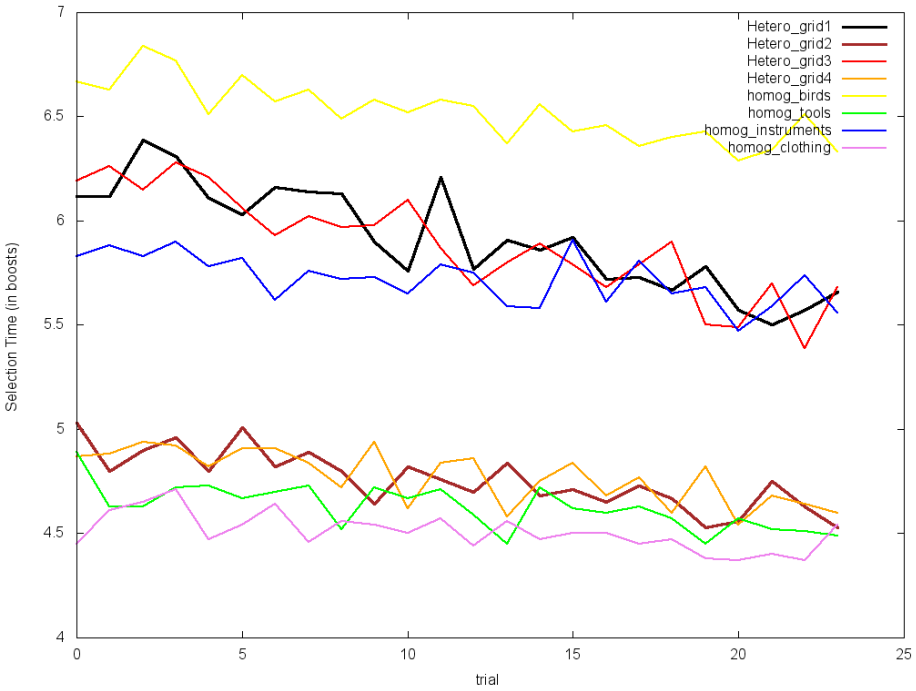

Figure 49 - Longevity Test, Epochs = 100, extended network

Comparing the four graphs reveals that the characteristic increase in boost counts over each cycle due to semantic interference is clearly diminishing over time. With (Epochs = 0) and (Epochs = 10) we see a clear step pattern. By the time we reach (Epochs = 100), however, this pattern has mostly disappeared. We can still identify the heterogeneous groups by sight, so there still is an overall cumulative semantic interference effect occurring – but within each trial, its effects are significantly reduced with respect to the initial testing phase. Thus, I might conclude that the amount of interference actually peaks much sooner than indicated by Figure 44; indeed, it seems to begin to decline shortly after the (Epochs = 10) point.

The graphs for the modified network are presented below for (Epochs = 0), (Epochs = 100), and (Epochs = 200) (Figures 50, 51, and 52):

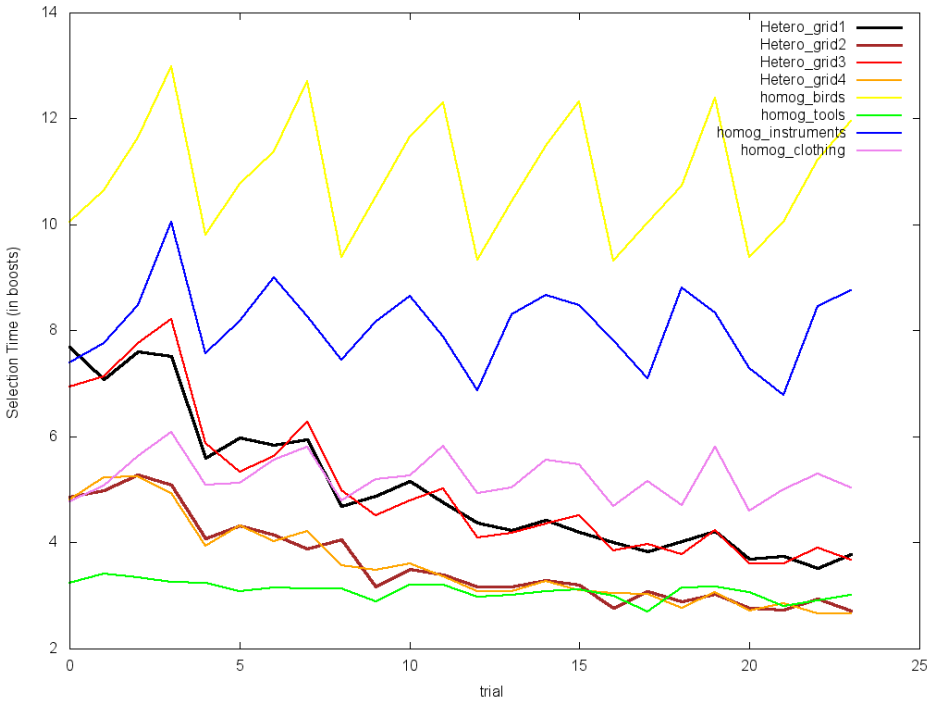

Figure 50 - Longevity Test, Epochs = 0, modified network

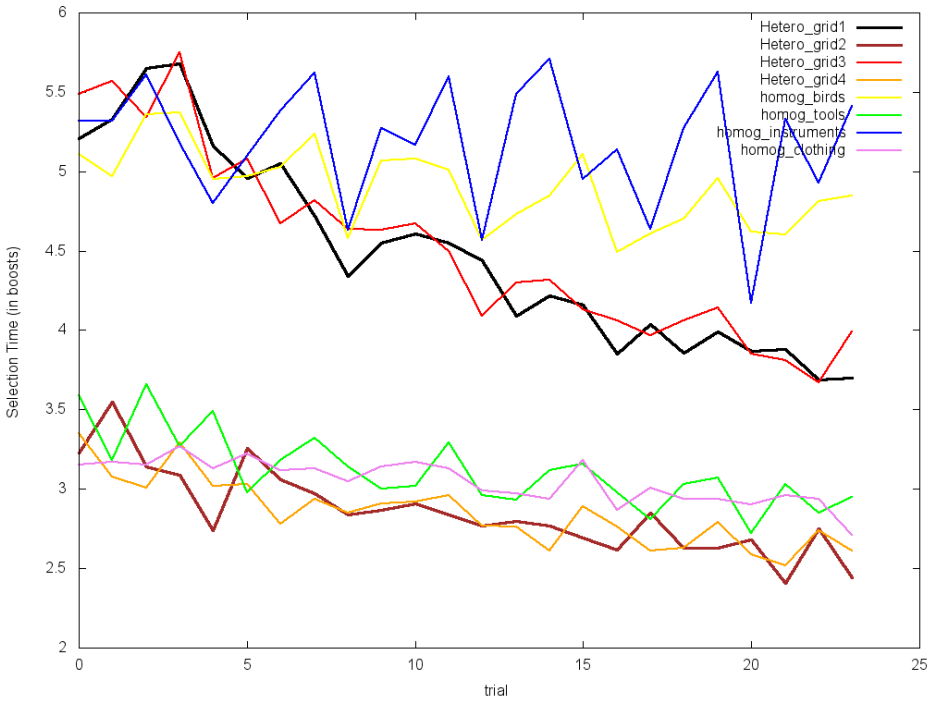

Figure 51 - Longevity Test, Epochs = 100, modified network

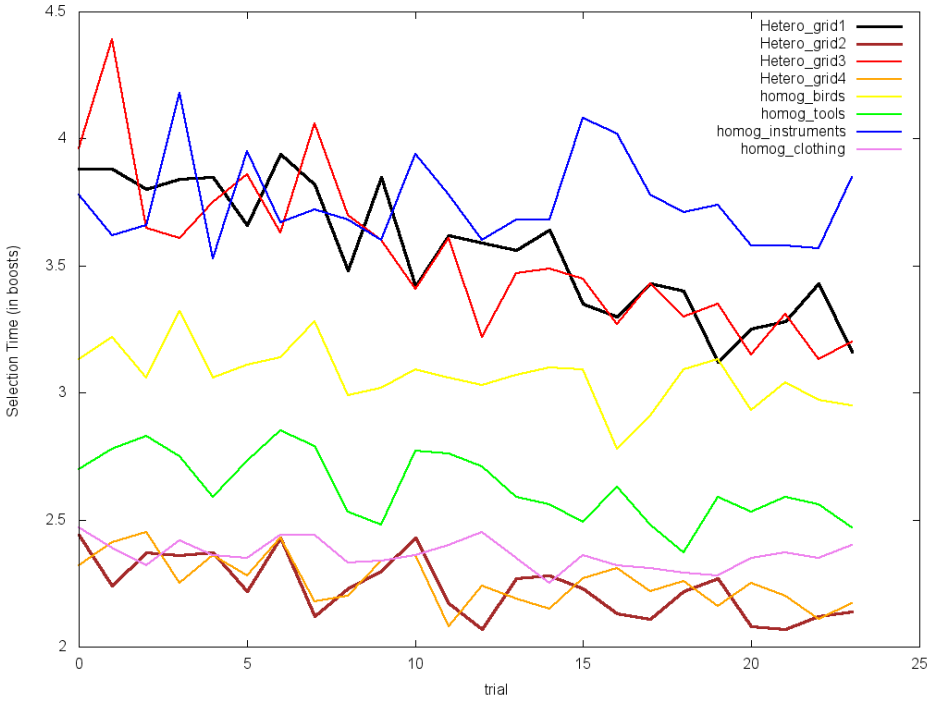

Figure 52 - Longevity Test, Epochs = 200, modified network

Comparing these three figures, we see that the characteristic step pattern caused by semantic interference does not dissipate until approximately the 200 epoch mark in the modified network. We also see the "birds" and "instruments" groups swap positions, as well as the "tools" and "clothing" groups. The reasons for this are unclear, but likely are due to the structure of the extraneous features described in Chapter 5.

Though the extended network only retains interference effects for 10 epochs, its total training period ***T*** was a mere 7 epochs. The modified network retains interference in memory for at least 200 epochs; however, its training period was approximately 700 epochs. Proportionally, it seems that the extended network is less sensitive to the passage of time – but ultimately, it is difficult to compare these two values meaningfully. The only conclusion I can safely draw is that both networks do retain the interference effects in memory for a significant period of time – at least as long as a significant fraction of the training period.

6: Final Remarks

6.1: Conclusions

In this thesis I have presented two extensions to the Oppenheim *et al.* computational model for cumulative semantic interference. Both extensions sought to generalize the original model into a framework over which more realistic simulations could be conducted. These models handle data derived from feature norms collected from human participants, and produce outputs which reflect a measure of semantic interference based on the semantic content of the data provided.

Furthermore, the modified model sought to include non-shared features of competitor outputs in the learning process, as well as features semantically related to activated features. It achieves this by using an algorithm that automatically detects and activates inputs that belong to both cases.

Both models succeeded in modeling the semantic interference effects seen in the model designed by Oppenheim *et al.* The extended model also correctly generalizes the effects seen to more complex datasets, and in all cases accurately reflects the internal structure of the data as interpreted by the metric functions, which were meant to serve as basic measures of semantic structure.

The modified model converged in all cases tested as well, but produced results different than those predicted by the designed metrics. Because the construction of the metrics was mathematically driven, and not designed based on principles of psychology, it is unclear which network is a more accurately predictor of semantic interference effects

as observed in humans. Only future research will corroborate these findings or determine that adjustments to this model are needed.

In examining the simulation results, I also found that as the space of learned words increased, features shared among *all* words became less and less relevant to selection times. This indicates that for sufficiently large networks, words that are composed of many features begin to be typified by a smaller subset of features that *most discriminate* them. I also see in the modified network a tendency to produce outputs that more closely correspond to the modified Chebyshev metric introduced in Section 5.2.3. This further supports the notion that larger, more complex networks tend to select words via a very small subset of strong features, rather than a larger subset of weaker features.

I found also in the modified network that the presence of extraneous homogeneous features actually benefits the learning mechanism. Learning is orders of magnitude faster for networks that contained many examples semantically related to their testing sets in their training sets. This suggests that improving the performance of the modified network is as simple as providing it with extra, semantically related examples to learn alongside the target examples. As the network grows larger, this procedure becomes less necessary, as these extraneous examples will naturally occur more frequently. This is partly the reason why I see such dramatically poor performance in the smallest modified network sizes for Simulation Group 1.

Finally, I see that both networks perform well in the presence of noise, though the modified network remained more sensitive than the extended network. I also see that the networks retain a state of interference even when presented with long interstitial periods

of random word activation, akin to results found in human experimentation. I also see that this interference eventually does decrease over time, as expected.

All of these properties make these networks useful for simulating large scale experiments that seek to measure memory retrieval time in response to various triggering conditions. The generalizations introduced allow them to be used on feature norm datasets designed for this purpose, unlike the original toy model designed by Oppenheim *et al*.

6.2: Future Work

There are two natural extensions to the models presented. The first of these concerns the modified network. As discussed in Section 4.3.4, the modified network's learning rule does not accurately calculate the gradient function, and thus does not modify the weights of the connections in the network as efficiently as it could. Modifying the learning rule by recalculating the expression for the gradient given the *activateSecondary* function would allow it to converge much more quickly. This in turn would allow it to be compared more easily to the results of the extended network.

The second natural extension is the addition of hidden layers in any or all of the networks presented. Examining the input vector using a multilayer convolutional network, for example, would be particularly interesting (Lee *et al.*, 2009). It might be the case that such a network forms natural "representative" neurons in the hidden layers that represent a hidden *semantic group*, which in turn might allow for the solution of remote association tests (RATs) over these data. These extensions were not pursued in the interest of time, however.

Another interesting problem, discovered while designing the simulation groups, was a method for algorithmically generating datasets with specified $\overline{D_G}$ and $\overline{D_A}$. This problem is actually an interesting optimization problem in computational geometry. Further exploring algorithms such as these would allow me to test these networks over a wider range of dataset configurations.

Finally, the modified and extended networks produce notably different results for complex datasets. Real-world experiments could be designed that have datasets analogous to the simulations run in Simulation Group 3, for example. By comparing the experimental results with the simulation results, I might be able to evaluate the hypotheses underlying the two networks' implementations. If the modified network demonstrates higher predictive power than the extended network, this would act as strong evidence for the "co-activation hypothesis" of O'Séaghdha *et al.* (2013), and would serve as a witness to many of the arguments presented in this thesis.

Bibliography

Brown, A. S. (1979). Priming effects in semantic memory retrieval processes. Journal of Experimental Psychology: Human Learning and Memory, 5(2), 65–77.

Duda, R., & Hart, P. (2001). Pattern classification (2nd ed.). New York: Wiley.

Frazer, A. K., O'Séaghdha, P. G., Munoz-Avila, H., & Roessler, N. (2014). Competitor Activation and Semantic Interference: Evidence from Combined Phonological and Semantic Similarity.

Haykin, S. (1999). Neural networks: A comprehensive foundation (2nd ed.). Delhi: Pearson Education.

Howard, D., Nickels, L., Coltheart, M., & Cole-Virtue, J. (2006). Cumulative semantic inhibition in picture naming: Experimental and computational studies. Cognition, 100(3), 464–482.

Lachman, R., & Lachman, J. L. (1980). Picture naming: Retrieval and activation of long-term memory. In New Directions in Memory and Aging (PLE: Memory): Proceedings of the George A. Talland Memorial Conference (p. 313). Psychology Press.

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Proceedings of the 26th Annual International Conference on Machine Learning (pp. 609-616). ACM.

McRae, Ken, et al. "Semantic feature production norms for a large set of living and nonliving things." Behavior research methods 37.4 (2005): 547-559.

Oppenheim, G. M., Dell, G. S., & Schwartz, M. F. (2010). The dark side of incremental learning: A model of cumulative semantic interference during lexical access in speech production. Cognition, 114(2), 227-252.

O'Séaghdha, P. G., Packer, D., Frazer, A.K., Preusse, K., Hatalis, K., Munoz-Avila, H., & Hupbach, A. (2013, November). Does mere co-activation drive semantic interference? Paper presented at the 54th annual meeting of the Psychonomic Society, Toronto, Ontario, Canada.

Preusse, K. & O'Seaghdha, P. G. (in preparation). Semantic interference: Insights from remote associates problem solving.

Wheeldon, L. R., & Monsell, S. (1994). Inhibition of spoken word production by priming a semantic competitor. Journal of Memory and Language, 33, 332–356.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. IRE WESCON convention record. New York: IRE (Reprinted in Anderson, J. A., & Rosenfeld, E. (Eds.) (1988). Neurocomputing: Foundations of research (pp. 123–134). Cambridge, MA: MIT Press.).

Vita

Tyler Seip was born October 8th, 1991, in Allentown, Pennsylvania, of James and Robin Seip. He received his Bachelor of Science in Computer Engineering at Lehigh University in May, 2014.