COGNITIVE PROGRAMS MEMORY

A FRAMEWORK FOR INTEGRATING EXECUTIVE CONTROL IN STAR

OMAR ABID

A THESIS SUBMITTED TO THE FACULTY OF
GRADUATE STUDIES IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL
ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

DECEMBER 2018

Abstract

Cognitive Programs are algorithms which guide the execution of visual tasks in the Selective Tuning (ST) model and its extension Selective Tuning Attentive Reference (STAR). However, no framework exists that allows for their rapid development, parameterization and execution.

In this thesis we propose that a basis set of elemental operations called Neural Primitives (NP) in conjunction with other control elements constitute a Cognitive Program. The Neural Primitives are biologically inspired computations that dictate the transformation functions from one representation to another and form the foundation of this thesis. Therefore a key theme here is using low level neural computations, the NPs to perform higher level cognitive functions that are required for task demands.

This thesis also introduces a database like structure called the Cognitive Programs Memory (CPM) that holds a set of predefined CPs which are accessible by the visual task executive (vTE) - a component of STAR with a major role of controlling task execution.

The CPs have been tested on the execution of three psychophysical experiments demonstrating how a sequence of operations can be used to complete visual tasks. The CPs have also been tested on two experiments demonstrating that the application of the NP operations result in qualitatively similar neural tuning curves to neurophysiological data.

Acknowledgments

I would like to thank my supervisor, John K. Tsotsos for his kindness, patience, insights and guidance from the very first day I joined the lab as a research assistant in Summer 2015 as well as his continual support and helpful suggestions throughout the development of this work.

I would also like to thank Thilo Womelsdorf for useful discussions and helpful suggestions. In addition, I would like to thank my committee member Mazyar Fallah for taking the time to review my work.

A special thanks goes to my colleagues and friends Raghavender Sahdev, Vassil Halatchev, Toni Kunic, Calden Wloka, Rakesh Sengupta and Oscar Gonzalez for providing words of encouragement, many laughs and helpful discussions for the development of this thesis.

Last, but certainly not least, I would like to thank my parents: Abid Hussain and Shahida Abid, and my siblings: Sumair , Zobia and Ayesha for their patience and support.

Table of Contents

List of Tables

List of Figures

List of Program Listings

List of Abbreviations

ACC	Anterior Cingulate Cortex
AS	Attentional Sample
BG	Basal Ganglia
CP	Cognitive Program
CPM	Cognitive Programs Memory
CPM-MD	Cognitive Programs Memory Methods Database
CPM-ME	Cognitive Programs Memory Method Elements
FBI	Feedback Inhibition
FC	Fixation Controller
FEF	Frontal Eye Field
FFI	Feedforward Inhibition
FOA	Focus of Attention
ISI	interstimulus interval
IT	Inferotemporal Cortex
LGN	Lateral Geniculate Nucleus
LIF	Leaky Integrate and Fire
LIP	Lateral Intraparietal Cortex
mLTM	methods Long Term Memory
NP	Neural Primitive
PAM	Peripheral Attentional Map
PC	Pyramidal Cell
PFC	Prefrontal Cortex
PING	Pyramidal Interneuron Gamma
PV	Parvalbumin cells

px	pixels
STAR	Selective Tuning Attentive Reference
tWM	Task Working Memory
vAE	Visual Attention Executive
VH	Visual Hierarchy
VR	Visual Routine
vTE	Visual Task Executive
vWM	Visual Working Memory

1 Introduction

1.1 Motivation

A person looks both ways before crossing the street and a complex activity of neural activations in the individual's brain occur during this process. How may we link such seemingly distinct concepts at two levels of abstraction together? More importantly, is it even possible to connect high level behavior to neural activations or is there just too much variability and complexity to solve such a problem? Clearly, this is an unsolved problem, however a good starting point to tie these two concepts together involves attention. Attention is defined as ".. a set of mechanisms that help tune and control the search processes inherent in perception and cognition" [82]. Indeed theories have been proposed that integrate attention at the level of behavior, networks, circuits and neurons [12].

Attention at the level of behavior is typically studied with psychophysical experiments in visual spatial orienting [56] and visual search [80] tasks. At the level of neural circuits and networks, a set of circuit motifs have been proposed to implement a diverse set of computations that may be important for solving visual tasks [92, 93]. In this thesis, we will attempt to bridge the gap between these levels of abstractions by implementing a set of algorithms for attention within the Selective Tuning and Attentive Reference (STAR) model [86].

There are two goals of this thesis. The first is to develop a database-like structure called the *Cognitive Programs Memory* (CPM) responsible for storing a set of algorithms that guide the system for executing a particular visual task. The algorithms used for accomplishing control are referred to as Cognitive Programs (CPs) [84] and stem from the concept of applying a sequence of operations until the goals of the task have been met, originally envisioned as Visual Routines by Ullman [88].

The second goal of this thesis is to propose a set of biologically inspired atomic operations called *Neural Primitives* (NPs). These act as the fundamental "building blocks", that when combined with some of the other elements in the CPM provide a set of useful computational functions that can be applied in a stepwise algorithmic manner.

CPs require a range of computations that are involved in pursuit of completing visual tasks. Some of these include providing top down control signals, selecting region of interest, dealing with covert and overt fixations, information routing, access to memory, matching to task and priming and coordination of bottom up and top down information [84]. The specific CPs executed are dependent on the processing strategy required by the visual tasks [85]. Furthermore, there may be other CP functions that are as of yet undiscovered or undefined which may complement some of the ones listed here. Although the set of computations CPs must implement has been thoroughly discussed and an implementation of CPs for playing video games has been demonstrated [30], there is no standardized method of reusing or adapting these algorithms for a variety of different visual tasks. Because of this, a user wishing to test a visual task in the STAR framework must know the precise task demands and implement the specific functionality required on a task by task basis making future research and development into the Selective Tuning model difficult and time consuming.

As a solution to these problems, we implement a set of CPs that are tailored to a specific visual processing strategy [85]. We propose that CPs are composed of two functional components: a) Queries and b) Actions. Queries are encoded as conditional statements and the results are used to make decisions while actions include access to representations (both read and write) via NPs, control signal and task specification and execution of an internal process. The underlying implementation of access to representations (write only) and a subset of control signals is done by a set of four computational operations referred to as *NPs*: a) Bias Control, b) Bias Feedback, c) Integration and d) Gating. Yes-no decision points are implemented either algorithmically or via the use of neural activation thresholds. Activation thresholds work on the basis of "triggering" a decision given a rule based criterion. Such rules are encoded as "if average activation of representation **X** is greater than 10.0 then return true, otherwise return false". The determination of control signals is also rule based as in the decision points and may also be influenced by the current state of a representation.

Figure 1.1 shows two components of the CPM - *Method Elements* (CPM-ME) and *Methods Database* (CPM-MD). *Neural Primitive (NP) Operations* can be integrated with *Base Methods* to assemble high level Cognitive Programs (*Stored Methods*). Here, a summary of each of these components is provided while a detailed description is left for subsequent sections.

1. *Method Elements*: Contains the necessary elements that are used to form other methods.
    (a) *Base Methods*: Composed of action and query methods.
    (b) *NP Operations*: Computational counterparts of biologically inspired neural correlates. Four operations, each with a distinct computation are proposed.
2. *Methods Database*: Stores assembled CP methods that are composed of the Method Elements. Here, we implement several methods that will be useful for demonstrating their functionality on different visual tasks. The Detection, Recognition, Categorization and Identification methods have been grouped together since they share similar Method Elements.

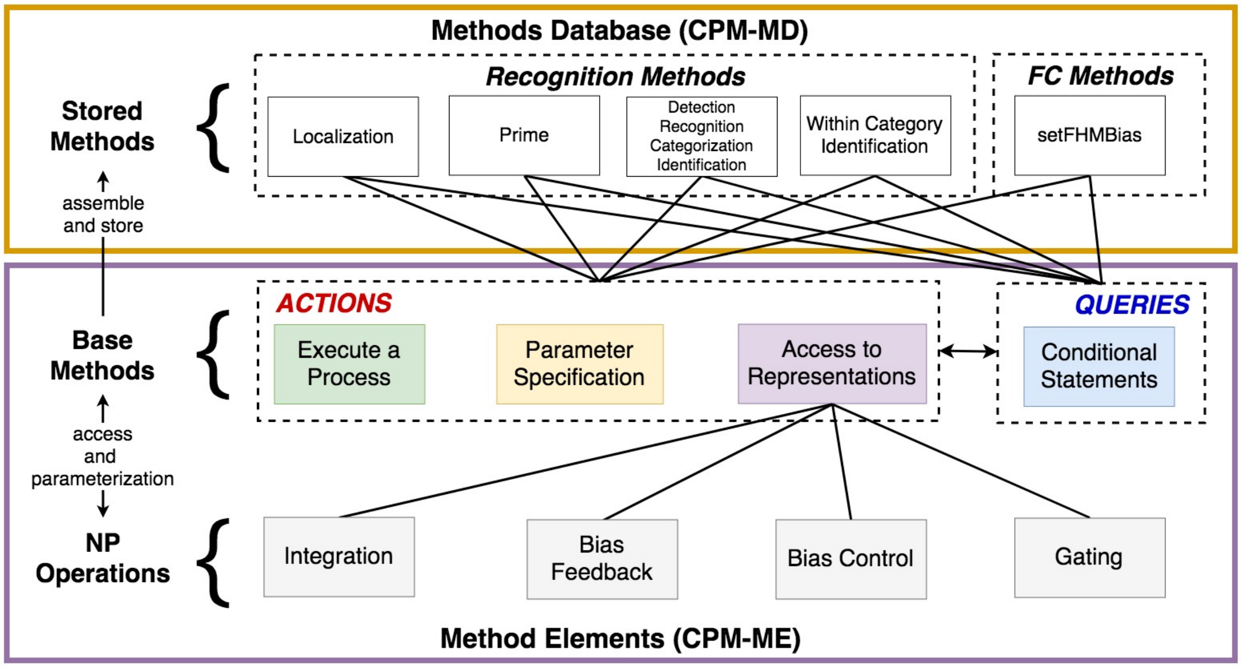


Figure 1.1: The Cognitive Programs Memory (CPM) is composed of the Method Elements (CPM-ME) and the Methods Database (CPM-MD). The CPM-ME uses low level NP operations in conjunction to a set of Base Methods to compose useful methods for task execution. Such methods are then stored into the CPM-MD. The arrow between Actions and Queries indicate that information captured from executing a process, parameter specifications and access to representations can be used for conditional statements.

The CPs which exist in the CPM operate on a fixed set of modules referred to as the visual attention executive (vAE), visual task executive (vTE), task working memory (tWM), visual working memory (vWM), fixation controller (FC) and visual hierarchy (VH). It is important to mention that of these modules, the vTE has recently been further developed [32] and is central to task execution, without which the CPM would not function.

To assess the CPM, we test on three psychophysical experiments employing distinct processing strategies to demonstrate a variety of different CP functionalities. In addition, we also test the CPs on a spatial and feature attention paradigm and show that the system is able to replicate qualitatively similar neural tuning curves to neurophysiological data.

1.2 Significance and Contributions

This thesis introduces the concept of Neural Primitives, the systematic construction of Cognitive Programs in part by these NPs and the introduction of the Cognitive Programs Memory.

The NPs build the foundations of the CPs as they dictate the mathematical function that is used to transform one representation into another. They are named accordingly since they are inspired by four distinct biologically plausible computations. Thus, a major contribution here is the formalization and justification of these operations.

The CPs have been defined previously, however, their implementation was not claimed to be biologically plausible. Thus, a second contribution is a neural realization of a subset of the total functionality of CPs.

Finally, the CPM provides a “storehouse” of the available CPs. In previous work, the visual task executive (vTE) was formalized with one of its functions being that it executes a sequence of CPs after retrieving them from the CPM [32]. However, at that time, the CPM had not been fully defined, thus a final contribution is the introduction of the CPM that allows for the storage and retrieval of CPs.

1.3 Thesis Outline

The **second chapter** of this thesis provides an overview of a breadth of literature including visual routines, the ST model, and the neurophysiological literature - including the Neural Correlates of attention and Neural Computational Elements. This provides a good starting point for integrating our *Neural Primitives* into the ST framework.

The **third chapter** is part one of two of the methods section. Here, a complete theoretical and mathematical definition of *Neural Primitives* is provided. In addition we explain the neural correlates and theoretical computations implemented by four Neural Primitives; Gating, Bias Control, Bias Feedback and Integration.

The **fourth chapter** is part two of two of the methods section wherein the *Cognitive Programs Memory* and *Cognitive Programs* are defined and formalized. Eight *Cognitive Programs* are also introduced including their algorithmic structure and biological plausibility.

The **fifth chapter** details the implementation of the *Neural Primitive* and *Cognitive Programs Memory* framework within the TarzaNN system.

The **sixth chapter** shows the evaluation of Cognitive Programs on three psychophysical visual attention experiments and two experiments showing the system is able to replicate both spatial and feature attention neural tuning curves.

The **seventh chapter** gives a summary of the thesis along with suggestions for future work.

In addition, three appendices are included providing details of the implementation of the framework. Appendix A provides a full list of functions and classes that were implemented for running the visual tasks described in Section 6. Appendix B provides additional experimental results of some of the internal processes executed by the Cognitive Programs. Finally, Appendix C lists the relevant experimental parameters used for executing all of our visual tasks as well as provide a complete STAR architecture with all the elements required for implementing and executing each Cognitive Program.

2 Literature Review

2.1 Visual Routines

How do humans carry out complex visual tasks such as making a sandwich [95]? One method may be to break it down to a sequence of simple operations that should be executed serially. For example, you may decide to first grab all the ingredients required and begin by applying peanut butter, followed by jam and finally putting the sandwich together. The underlying "sequence of operations" are a key theme in this thesis originally stemming from Ullman's Visual Routines (VR).

Ullman proposed a theory that visual tasks can be solved by reducing the required processes to a set of elemental operations that may be parameterized for the given task at hand. He presents a strategy for extracting shape and spatial relations that are essential to completing a visual task by applying a sequence of transformations on representations [88]. Furthermore, inspired by the work of Marr's theory of vision [38], he suggested that vision can be broken down into a two stage process. First, a bottom up base representation is created from the incoming visual data, then various operations are applied sequentially to the base representation to solve a particular visual task. The basic components of the theory are highlighted below.

1. *Base Representation*: is assumed to be derived automatically and assumed to describe local image properties (see Marr's 2.5D Sketch [38]) such as the color, orientation, motion and depth information.

2. *Incremental Representation*: is the result of applying visual routines to the base representation.

3. *Atomic Operations*: are the unit operations that can be applied to a location within the base or intermediate representation. These are the single irreducible units in a larger system [76]. The operations are listed below:

   (a) *Shift of Processing Focus*: Shift the region the atomic operation is to be applied.

   (b) *Indexing*: Defining next target for focusing attention.

   (c) *Marking*: Storing locations for future use.

   (d) *Boundary Tracing*: Tracing a region.

4. *Assembly*: VRs are assembled by sequencing elemental operations.

5. *Execution*: VRs require visual attention for execution, the details of the implementation are not discussed.

6. *Storage*: VRs may be stored in a "skeleton" framework and can be applied at different spatial locations. In addition, new routines may be assembled and stored given task demands.

Notice here that the atomic operations work on the principle of extracting subsets of the incoming data for future processing. For example, the *indexing* and *shift of processing focus* operations are responsible for defining the next location of interest and moving to that location, respectively. The *marking* operation serves as a form of memory if locations are required for use at a later stage of processing. *Boundary tracing* performs some tracing computation within the selected region. This idea of applying operations within a region allows us to limit the processing to a small attended region.

Cavanagh 2004 [16] divided routines into three levels: visual routines, attention routines and cognitive routines after taking inspiration from Ullman's VRs. He described Visual Routines as automated processes that are inaccessible to awareness. Attention routines on the other hand are consciously initiated by setting a goal or the selection of a target (find the red item) with the intermediate steps being visual routines. Finally, cognitive routines contain multiple steps involving action, memory, vision and other senses that may contain attention and visual routines as part of their intermediate steps. Here, cognitive routines may be used for high level tasks such as driving home or baking a cake. Table 2.1 summarizes this below.

	Visual Routines	**Attention Routines**	**Cognitive Routines**
Description	Automatic	Voluntary initiation. Reportable output. No intermediate, accessible states.	Multiple steps with intermediate steps available for access.
Examples	Grouping, light constancy, shape recognition, pictorial cues	Set selection criteria, spatial & temporal relations, tracking	Counting, cooking

Table 2.1: Three levels of visual routines arranged in a hierarchy of complexity from left to right. Whereas visual routines are used to extract low level visual features, attention and cognitive routines make use of visual routines to solve complex visual tasks. Reproduced from [16].

Recent neurophysiological work supports the concept of Ullman's VRs with literature suggesting that serial cognitive tasks may be implemented by the interaction of brain networks spanning several cortical areas [60]. In one experiment, monkeys were asked to do curve tracing and visual search in different orders and recordings were made in area V1 of the visual cortex. The authors monitored the precise time course of these operations and found that the ordering of neural activations is dependent on the ordering of the tracing and visual search operations.

This temporal ordering of cognitive operations provides evidence of encoding complex visual tasks in the primary visual cortex via a set of elemental operations [47]. Similar findings have been found by other authors [61].

Several other works use VRs to model complex visual tasks. For example Yi and Ballard [95] model human behavior on visuomotor tasks of making a sandwich or pouring coffee using a dynamic Bayes network and show how the steps implemented in the network are akin to visual routines.

Roelfsema [60] proposed two categories of operations: binding and maintenance operators based on neurophysiological findings and Ullman's VRs. Binding operators are responsible for establishing groupings that are not available in the base representation while maintenance operations store intermediate results that may be useful for use at a later point. The proposed elemental operations are listed in Table 2.2. To demonstrate these operations, the author showed how they could be used to find your way on a map. The first process is defining your current position on the map and the destination position you wish to reach. Then, through a process of applying visual search, tracing, movement (motor actions), cuing and suppression operations, one can map out the route required to reach their final destination (see Box 2 in [60]).

Binding	**Maintenance**	**Other Operations**
searching, cuing, tracing, region filling, associating	Working memory, suppression	Matching, Motor actions

Table 2.2: A summary of the elemental operations proposed by Roelfsema [60]

Another model that uses the idea of sequencing operations proposes that three classes of neurons: a) *evidence accumulation neurons*, b) *action neurons* and c) *memory maintenance neurons* may interact and be combined sequentially to form mental programs or visual routines [97]. *Evidence accumulation neurons*, also called "ramping neurons", integrate evidence over time to improve signal-to-noise ratio, and typically result in the firing of a production rule when the neurons activation reaches some threshold. *Action neurons* participate in a winner take all (WTA) competition for a selection of one action among others. These neurons are typically used to encode a production system that allows for the implementation of *IF-THEN* like rules. In [96], the authors suggest that a "chain" firing of these action neurons may be one mechanism the brain uses to sequence serial operations as well as dealing with noise propagation issues that would be inherent in analog systems like the neuron. *Memory maintenance neurons* maintain states of information encoded at previous times steps that may be useful for future operations. Such a model has been used to show how arithmetic and more complex visual tasks can be encoded using these three classes of neurons thus highlighting the biological plausibility of the framework [96].

In this section we have highlighted neurophysiological support for VRs as well as several models that implement them for solving cognitive tasks. However, there is a tremendous amount of literature that is beyond the scope of this review. For a more comprehensive look at various implementations of visual routines and their applications, please refer to [30].

2.2 ST Model

In the previous section we had looked at Ullman's VRs as a theory for explaining how complex visual tasks may be solved by a sequence of operations. However, recent neurophysiological literature suggests that a base representation formed from a purely bottom up stimulus is not enough to gather all the knowledge about the scene or to provide sufficient details about objects within the scene. Furthermore, the conceptualization of VRs is based on attention being purely spatial that precedes interpretation; such a view is also no longer supported by modern knowledge [15]. Instead, an interaction between bottom up and top down processes is required [3, 12, 44] to explain the breadth of processes involved.

Visual attention is defined as ".. a set of mechanisms that help tune and control the search processes inherent in perception and cognition" [82] and is central to several models of visual attention but why is it important? The answer to this comes from the limited processing capabilities of the brain [83].

Among the many models of visual attention, we review the selective tuning (ST) model, with the foundation of the framework rooted in advancements in the neurophysiological literature making it a biologically plausible computational model [82]. Here, among the many differences this model has with respect to Ullman's feedforward formation of base and incremental representations include recurrent processing that necessitates top down processing, and the hierarchical nature of the visual cortex. These differences suggest that a simple base representation does not carry all the information we require and instead a series of processing steps occur before the representation is available for applying some operations. A detailed discussion is provided by Kruijne and Tsotsos [31].

In this section we will look at CPs; a modernized version of Ullman's VRs that are responsible to complete a visual task [84]. We will then review the selective tuning and attentive reference (STAR) model; an extension of STAR to allow for executive control with a set of algorithms; Cognitive Programs [86].

2.2.1 Cognitive Programs

CPs are the algorithms that guide the execution of visual tasks and are a modernized version of Ullman's Visual Routines [84]. CPs require a range of computations such as providing top down control signals, selecting region of interest, dealing with covert and overt fixations, information routing, access to memory, matching to task, priming and coordination of bottom up and top down information [84].

The specific CPs that are required to be executed depend on the task requirements. For example, a simple discrimination task would only require a single feedforward pass through the VH while an identification task would require recurrent processing. Tsotsos et al. [85] identify several tasks, the approximate time required for processing and the attentional processes taking place during its execution. Furthermore, the authors identify the binding processes associated with each task. The implication of this indicates that a small subset of computational strategies can be used to execute a variety of different visual tasks.

In light of this, Figure 2.1 shows a variety of visual search tasks that overlap in the algorithms they require. The color coding in the figure indicates which CP the command belongs to. Notice how the CPs rely on other CPs given task demands.

CPs are also influenced by the idea of *storage* and *execution* in Ullman's theory of VRs. A stored CP, as a *method* must first be fetched from the CPM. This *method* acts as a "skeleton" algorithm that is task independent and cannot be executed without providing information about the task. A *script*; an executable version of a tuned *method* is formed when a task specification is provided. For example, a specification of the color or location of the target for a given *method* results in the formation of a *script*.

Despite this, there is no standardized method of reusing or adapting these algorithms for a variety of different visual tasks. Thus, a user wishing to test a visual task in the STAR framework must know the precise task demands and implement the specific functionality required on a task by task basis making future research and development into the Selective Tuning model difficult and time consuming. Part of the aim of this thesis is to formalize and implement CPs within the CPM framework in the TarzaNN software [64].

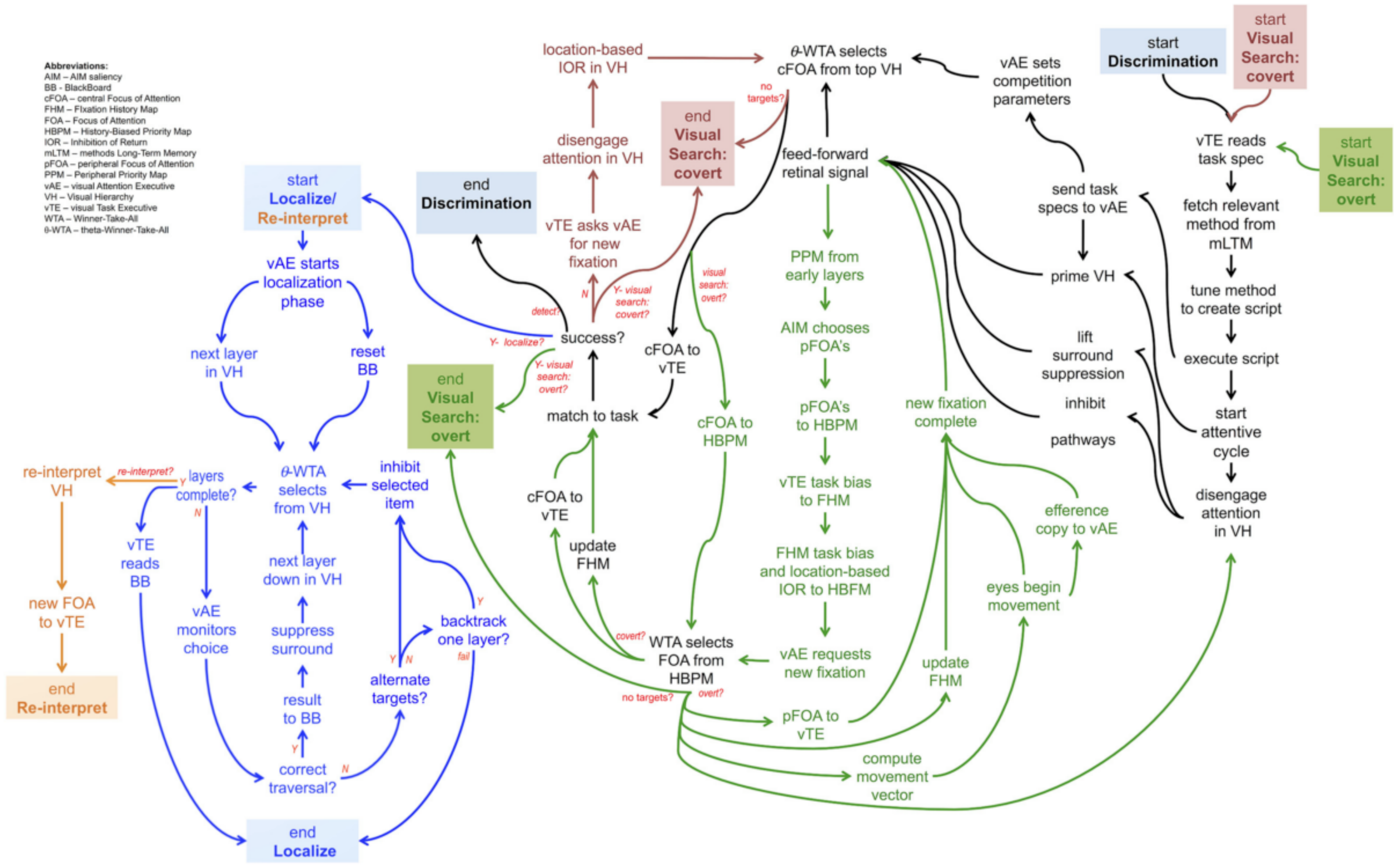


Figure 2.1: A graphical depiction of multiple Cognitive Programs that are sequenced together to solve complex visual tasks. The color coding indicates which Cognitive Program the commands belong to. Reproduced from [84].

2.2.2 STAR

In this section we will review the major components of STAR as highlighted in Figure 2.2.

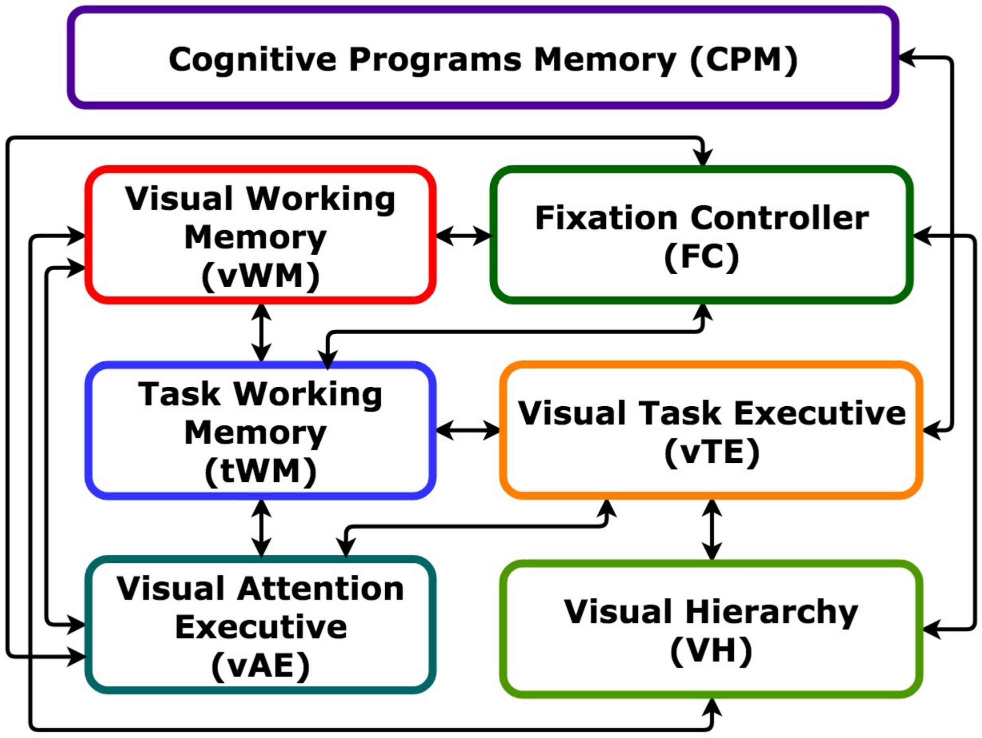

Cognitive Programs Memory (CPM)
Visual Working Memory (vWM)
Fixation Controller (FC)
Task Working Memory (tWM)
Visual Task Executive (vTE)
Visual Attention Executive (vAE)
Visual Hierarchy (VH)

Figure 2.2: The components of STAR with corresponding arrows indicating the flow of information between modules.

Visual Hierarchy The visual hierarchy (VH) is the most well developed part of the ST model beginning with the processing of some visual input signal and a $\theta - WTA$ (winner-take-all) selection mechanism for the required features or locations is applied at each layer of the hierarchy. Unlike a standard WTA algorithm in which the neuron with the highest activation is selected while all others are suppressed, $\theta - WTA$ algorithm relaxes this requirement such that several neurons with firing rates within some threshold θ of one another are selected at any one time.

The processing of stimuli through the VH occurs in distinct stages. First, the hierarchy is primed for a location or feature of interest before stimulus onset. Next, a feedforward pass occurs followed by a $\theta - WTA$ selection process at the top of the VH. If further information about the stimulus is required that is not available at the top most layer of the VH, recurrent top-down localization suppresses all the neurons that do not contribute to the winner at the very top of the VH at each layer. Following this, a reinterpretation of the information occurs through a feedforward pass with the suppression mechanisms in place.

The focus of attention (FOA) created after recurrent processing contains the neurons that contributed to the winner at the top of the VH. All the information available in the FOA across all layers of the visual hierarchy is called the attentional sample (AS) while a partial attentional sample (pAS) may contain only a subset of layers. The AS may be stored into the visual working memory (vWM) or task working memory (tWM) for use in later stages.

Visual Working Memory The visual working memory (vWM) stores a representation of the VH (as a Partial P-Lattice) in an AS. The AS stored here can be used to extract location or feature information about the target or serve as the basis for priming the VH. The vWM also contains the Fixation History Map (FHM) intended to bias against revisiting previously seen locations [29] which stores the last few fixations. Thus, it interacts directly with the fixation controller.

Task Working Memory The task working memory (tWM) may be used to store a sequence of attentional samples, fixations, selections, scripts and script progress. In other words, any relevant information during task execution is stored here.

Fixation Controller Attention can be directed to a stimulus with either *covert* or *overt* attention. *Covert attention* refers to attending to a stimulus without any eye movements. *Overt attention* occurs when attending to a stimulus by pointing their gaze at the stimulus, thus requiring eye movements. The Fixation Controller (FC) guides these processes by taking into account (a) the peripheral visual field ($> 10°$) derived from early visual representations, (b) the Peripheral Attentional Map (PAM) and the central visual field from foveal representations and (c) the Central Attentional Map (CAM), to form a priority map. A separate representation that is larger than the visual field, the fixation history map (FHM), contains recent fixations with task specific biases that may be in place. The FHM is combined with the priority map to influence the FCs next eye movements. For an in depth look at the FC strategy, please see [86].

Visual Task Executive The visual task executive (vTE) is responsible for constructing, executing and monitoring a script. A recent development of the vTE is the addition of a Cognitive Program Compiler (CPC) [32]. A CPC takes human understandable task description and translates this into machine code as a task script. The task script contains a sequence of commands including the execution of CPs after retrieval from the CPM, decision points and other functions relevant for completing a visual task. For example, a task description of "find 'X' in the stream of letters" could be translated by setting the appropriate parameters and deciding the relevant CPs to execute sequentially. The CPC provides an excellent strategy for rapidly executing visual tasks by defining the sequence of commands required for task completion. Thus, the vTE plays a fundamental role in the architecture indicated by its connectivity to other major components of STAR: vAE, CPM, tWM and vWM.

Visual Attention Executive The visual attention executive (vAE) generates the necessary control signals for completing visual tasks using the task specification provided by the vTE. This is most prominent in the VH but also relevant in other areas of this model including the FC, vWM and tWM [84]. Notice that the vAE is connected to every module in STAR except for the CPM.

Cognitive Programs Memory The CPM stores prebuilt CPs that can be retrieved, parameterized and executed by the vTE. This thesis formalizes the CPM and consists of the following components:

- *Method Elements*: Contains the necessary elements that are used to form other methods.
  - *Base Methods*: Composed of action and queries methods.
  - *NP Operations*: Computational counterparts of biologically inspired neural correlates. Four operations, each with a distinct computation are proposed.
- *Methods Database*: Stores assembled CP methods that are composed of the Method Elements. Here, we implement several methods that will be useful for demonstrating their functionality on different visual tasks. The Detection, Recognition, Categorization and Identification methods have been grouped together since they share similar Method Elements.

ST Microcircuit The Selective Tuning (ST) circuit consists of interpretive, bias, gating and gating control neurons. Each ensemble of neurons as shown in Figure 2.3 computes a single visual quantity (such as a feature or object) competing with other ensemble of neurons at the same spatial location for the representation of the visual quantity.

Interpretive neurons also known as feature detecting neurons, receive feedforward input from other areas with inputs arriving in layer IV while providing an output to layer V/VI. Bias neurons provide top-down guidance for visual processing such as parameterizations for locations, regions or features. A key component of the ST microcircuit is the gating and gating control neurons in charge for implementing the selection mechanism and tracing those neurons down the visual hierarchy to their source forming the attentional beam.

$$\frac{dr}{dt} = \frac{1}{\tau}\left[-r + B * S\left(\sum_{k\epsilon\uparrow}\gamma_k g_k e_k + \sum_{h\epsilon\leftrightarrow^a} g_h e_h\right)\right] \tag{2.1}$$

$$B(t) = \min_{\beta\epsilon\downarrow^b} B_\beta(t) \tag{2.2}$$

Where g_k and e_k are the feedforward weights and the activation of neuron k, respectively. Likewise, g_h and e_h are the lateral weights and activation of neuron h, respectively. $\uparrow$ represents the set of feedforward connections to the neuron from all sources, $\leftrightarrow$ represents the local connections to a neuron and $\downarrow^b$ is the set of bias units making feedback connections to the neuron and γ_k is the gating neuron whose value falls as the competition proceeds to reflect the branch and bound pruning process. In addition, the following constraints are placed on the parameters. For a more detailed description of the ST microcircuit refer to [63, 82].

1. $0.0 \leq B \leq 1.0$
2. $-1.0 \leq g_h \leq 1.0$
3. $0 \leq \gamma_k \leq 1.0$
4. $-1.0 \leq g_k \leq 1.0$

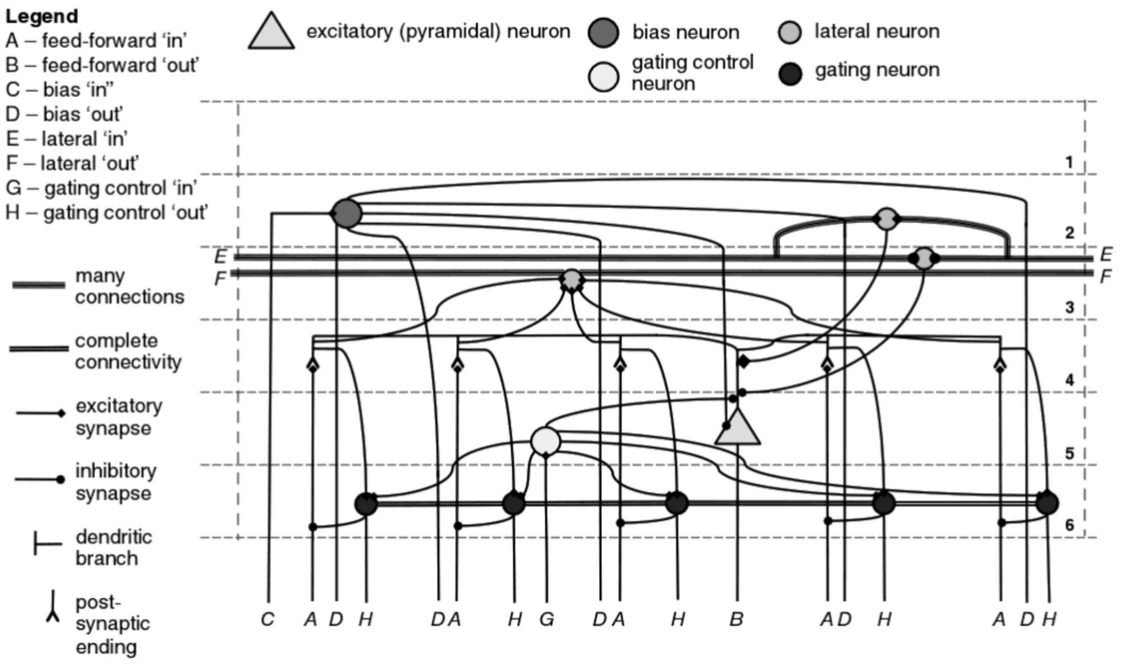


Figure 2.3: The complete ST microcircuit represented by equation 2.1. Reproduced from [82].

2.3 Neurophysiological Literature

One of the key questions in neuroscience is how does neural activity lead to high level behavior that we see in everyday life such as walking and talking. Of course, this is still an unanswered question, however many avenues of research have tried to correlate neural activity with high level behavior [12]. In doing so, researchers have identified that certain brain regions are associated with specialized functions [3, 92]. Others have identified a set of recurring circuit motifs that may encode simple computational functions [87, 93].

The aim of this portion of the literature review is to understand the interaction between the sensory information, an organism's goals, motivations and the control mechanisms that underlie their processing. In order to do this, we divide the review into three sections. First, we discuss the brain anatomy followed by a review of the neural correlates and circuits involved in attention.

2.3.1 Anatomy

In this section we discuss the basic anatomy of neurons, a cortical hypercolumn, important functional brain regions and the kinds of connectivity that govern their interactions.

Neuron The neuron is the basic building block of the nervous system and is responsible for receiving and transmitting nerve impulses [28] from dendrites and axons, respectively.

Dendrites receive input from a variety of different neural axons as an action potential in an all-or-nothing firing fashion. The incoming action potential to the dendrite reaches synapses located on the dendritic surface allowing the action potential to be transmitted to the connecting neuron. On the other hand, axons carry impulses away from the cell body onto other neurons. The soma (see Figure 2.4) is referred to as the cell body and varies considerably from neuron to neuron.

The neurons in the central nervous system can be classified based on the length of their projections. Pyramidal cells (PC) and satellite cells are those with long and short axons, respectively, resulting in making long range or short range local connections respectively.

Neurons may also be classified in terms of the neurotransmitters they release. The Pyramidal Cell (PC) mentioned earlier are typically excitatory [10] while satellite cells such as interneurons are inhibitory [42].

Cortical Hypercolumn The cortical hypercolumn was first described in Hubel and Wiesel's studies on the primary visual cortex [27] with the notion that these columns are highly repetitive

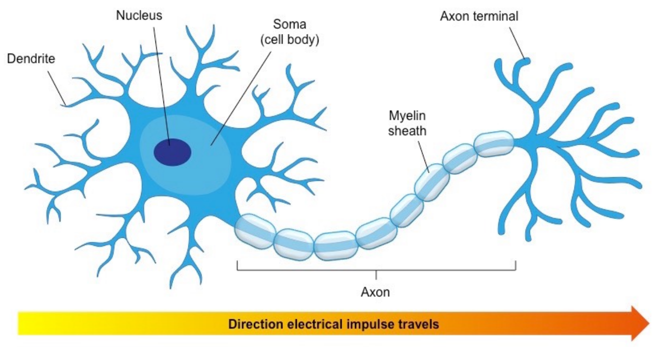


Figure 2.4: Components of a Neuron. Digital image. *Bioninja.* Web. 29 November 2017. <ib.bioninja.com.au>.

modules that function in brain organization. However, considerable variability of the cortical column between different areas and species has also been found [14]. Nonetheless, a "basic outline" of the organization of the neocortex is provided here.

1. **Layer I:** Layer I is sometimes referred to as an acellular layer since it contains less than 0.5% of all cells in a cortical column [5]. This layer is known to contain almost all inhibitory cells and provide strong mono-synaptic inhibition to LII/LIII cells.

2. **Layer II/III:** This layer contains pyramidal cells which receive input from excitatory neurons in LIV and from local interneurons while projecting to LV/LVI. Reciprocal connections also exist between LII/III to LV [10].

3. **Layer IV:** This layer contains a variety of excitatory and inhibitory neurons that receive most of the feedforward thalamic input [10] and project their axons mostly to interneurons in layers I and II providing feedforward inhibition within the same cortical column. However, excitatory projections to LII and LIII are also present.

4. **Layer V/VI:** A variety of excitatory (PCs) and inhibitory (PV^+,SOM) neurons exist in Layer V [49]. The PCs in this layer provide long-range projections to other cortical and sub-cortical structures thus playing a key role in top down control to other cortical areas. Specifically, LVI receives and projects connections to the thalamus [10] functioning as both an input and output. Short range efferent and afferent connections with all layers of the same cortical column are also present [37] but particularly to LI.

Visual Cortex Information processing begins when light enters the eye and projects onto the retina. Then, through a series of pathways information reaches the Lateral Geniculate Nucleus (LGN). Information from the LGN is then transmitted to the primary visual cortex (V1) which then transmit to higher order visual cortical areas [9] for more complex image processing (see

Figure 2.5). In this section we will provide a high level description and functions of some of the important visual pathways and areas that are directly relevant to this thesis.

The primary visual cortex is the largest cortical area in primates and is also referred to as V1 and the striate cortex due to its appearance. V1 provides a retinotopic map of the field of vision. However, it is not uniformly distributed across the entire visual field. Instead, the densest part of the retina, the fovea take up about 25% of the representation in V1 [9]. Neurons in this area are sensitive to simple features such as visual orientations, spatial frequencies and color.

The primary visual cortex projects to the extrastriate cortex consisting of V2,V3,V4, inferotemporal cortex (IT), middle temporal area (MT) and the posterior-parietal cortex (PP) [9]. The flow of information into these areas is thought to be segregated into two streams referred to as the ventral stream (or "what" pathway) and the dorsal stream (or "where" pathway). The ventral stream is associated with object detection and form representation and includes areas V1,V2,V4 and IT. As such, it is sensitive to high spatial frequencies and detailed image processing. The dorsal stream is associated with where objects may be located, their motion and depth and include areas V1, V2, V3, MT and PP [9] amongst several others. The dorsal stream is less sensitive to high spatial frequencies but more to high temporal frequencies allowing it to specialize in motion processing [51].

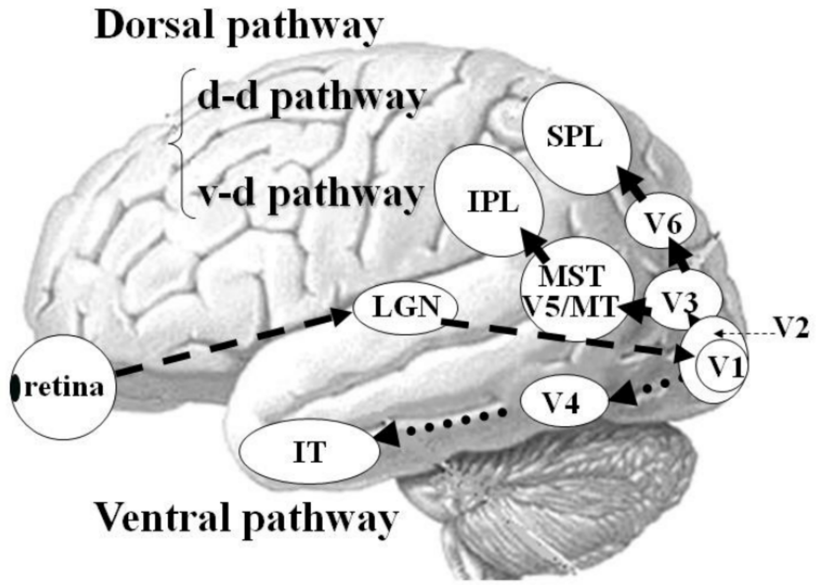


Figure 2.5: The dorsal ("what") and ventral ('where") pathways project to areas V2, V4, IT and V2, V3, MT and PP respectively. Reproduced from [94].

Connectivity There are three kinds of connectivity that are typically mentioned in the neurophysiological literature which describe how neurons interact with each other. The importance of the difference lies in the information they provide individually and collectively. The principle of segregation and integration states that these three types of connectivity can be useful when thinking about segregated networks (anatomical connectivity) that can integrate information across a variety of different sources (effective and functional connectivity) [79].

1. *Functional Connectivity*: Describes the statistical dependencies between segregated brain networks or regions and is shown to correlate with behavior in different tasks [25]. For example, the default mode network (DMN) is one that is active when an individual is in a resting alert state and not focused on any particular task [11].

2. *Effective Connectivity*: Describes the causal interactions that occur in brain networks such as a signal change appearing in one area directly correlating with a signal change in another area. Effective connectivity measures these time and task dependent patterns of activation during different cognitive tasks [25].

3. *Anatomical Connectivity*: Describes the physical connections between brain regions thus constraining the possible computations that may be performed [7].

2.3.2 Neural Correlates of Attention

Brain Networks Which brain areas are involved in perception, attention and coordination of information? Several theories for the “source” of attention mechanisms have been proposed with the general notion that attentional mechanisms may be described by the integration of feedforward perceptual bottom up information with top down attention that reflects the organisms behaviors, goals, motivations and current state [3, 12, 44]. However, this terminology of “top-down” attention may be misleading as it suggests that there is a single control mechanism. Instead, endogenous control of attention better reflects these internal processes as it suggests that multiple cortical and subcortical areas work together to give rise to behavior as supported by numerous neurophysiological studies [92].

Attentional selection is mediated by interactions between the frontal, parietal, temporal and occipital cortex in conjunction with the thalamus and mid brain areas [12, 92]. Notable areas in the fronto-parietal network that are known to be activated when subjects attend to a location in space [12] include the superior parietal lobe (SPL), the intraparietal sulcus (IPS), the FEF and the supplementary eye field (SEF). Interestingly, the interaction of the fronto-parietal network is well studied [44] and provides valuable insights of how information flows between these two areas under different conditions. For example, the lateral intraparietal area (LIP) in the parietal cortex contains saliency maps [2], however, the frontal cortex is also known to contain saliency maps but with a larger latency of activation with respect to stimulus presentation [44]. This leads to the idea that bottom up capturing of attention with a salient stimulus results in activation of the LIP followed by the frontal eye field (FEF) in the parietal and frontal networks respectively. In contrast, internally directed attention by the memory of a target stimulus results in activation first in the frontal network than in the parietal network [13]. The full set of networks involved in this frontal-parietal network interaction include V1 -> V2 -> V4 -> MT -> LIP -> LPFC -> FEF [44].

With regard to top down or endogenous attention, the PFC is thought to play a pivotal role in cognitive control including the ability to carry out new or goal directed behavior, motor attention, short term memory, inhibition of irrelevant information, gating of information, sequencing tasks, planning and problem solving [72]. The PFC is able to have such a diverse role in cognitive control in part by its extensive reciprocal connections to major cortical and subcortical brain regions. The basal ganglia is a subcortical brain region that also plays an important role in executive functions, memory processing and computing gating functions via a disinhibition mechanism on its output nuclei [33]. The overlap in the computations the basal ganglia and the PFC perform is complemented by the existence of both indirect connections through the thalamus and direct innervations between these areas. In light of this, several others have suggested that the basal ganglia acts as a gating mechanism for representations in the PFC by deciding when they should be maintained or updated [53].

One of the ways we know how bottom up and top down influences of attention interact is through microstimulation and lesion studies in macaque monkeys. Stimulation in LIP results in a bias toward the corresponding location in the visual field during a visual search task [3]. Sub and supra-threshold stimulation in the same area has also lead to covert and overt shifts of attention respectively [3]. In FEF, stimulation of one area results in a corresponding change of neural activations in area V4 [46] while suppressing irrelevant stimulus representations. Another study shows that deactivation of FEF interrupts planned saccades yet has no effect on bottom-up stimulus detection [35]. Furthermore, PFC lesions in macaque monkeys result in a deficit of attentional shifts but has no effect when attention is grabbed by a salient stimulus [62].

In summary, there is a tremendous amount of complexity in the interaction of these brain networks and for a more detailed review, see [3, 12, 44].

Neural Synchrony The patterns of neural activations that occur across different neurons are shown to be dependent on the information being processed and task goals [93] and thus may reflect the underlying mechanisms of network interactions and the selection processes of neural representations [44].

Synchronous inputs arriving at downstream neurons have been shown to have a synergistic effect on the neural firing rate [67] and such a mechanism may be involved in improving the signal to noise ratio (SNR) while reducing the total number of spikes required to represent a stimulus [77]. A consequence of this property is that synchronized neurons encode more information than unsynchronized neurons as evident in both the prefrontal cortex and the parietal cortex [44].

An interesting property is the different frequency at which neurons synchronize given task demands. For example, neurons in area V4 show an increase in gamma or alpha band activity if the neurons represent an attended or unattended stimulus respectively [24]. This segregation is thought to reflect a mechanism in which the signal (or attended stimulus) is separated from the noise (or unattended stimulus) [17].

Because of the nature of these neural oscillations, it is suggested that synchrony between different brain areas may be a mechanism in which they interact by changing their effective connectivity. This synchrony is seen in prefrontal, parietal and visual cortical areas [13] but also in subcortical structures such as the thalamus [65]. A notable neural signature of visual attention shows a high frequency (gamma) synchronization of cortical areas with a low frequency (alpha and beta) synchronization between the cortex and thalamus respectively [66]. Having said this, to date there is no causal link between the oscillatory processes and the functions that have been attributed to them.

2.3.3 Neural Computational Elements

Overview Up until now we have considered neurophysiological literature pertaining to the anatomy of excitatory and inhibitory neurons, the structure of a cortical column and the different brain areas involved in attention and decision making. The aim of this section is to outline how neurons or groups of neurons, *neural circuits*, may implement neural computational elements. More specifically, with the inspiration of Ullman's Visual Routines, we wish to study how such operations may be implemented within the brain with the following question in mind: "*Can we represent operations as a set of biologically plausible computations?*".

In the context of neural computations, several lines of work suggest that there exist some fundamental "building blocks" of the brain that may be used to form "computational units". As this thesis is aimed at proposing and implementing a basis set of operations for our Cognitive Programs, these neural computations guide and constrain the possible operations we may have. In the following sections, we begin by outlining some high level computations that may be implemented by neurons and neural circuits followed by the introduction of important circuit motifs that underlie such computations and finally provide a summary of the findings.

Computations Arguably, the neuron may be the most fundamental computational element. Single neurons may reflect the evidence towards a decision [8, 68], be sensitive to a feature [58] (such as a color or orientation), a task property [69], representations of sequences [18, 61, 91] while a complex interaction of neural circuits may reflect short term memory maintenance [93].

Neural circuits have been shown to implement a variety of different computations such as context, phase or synchrony dependent gating, gain control and integration [93]. Arithmetic functions such as addition, subtraction, multiplication and division are also important computations reflected in input-output neural response curves [73]. An important arithmetic computation that has been found in a variety of cortical and subcortical areas is that of divisive gain control and normalization (implementing a division/multiplication of the input-output response curves) [87], with interneurons playing a vital role in this computation [12]. In addition, single neurons have been shown to compute nonlinear functions possibly due to an interaction of synaptic plasticity, synaptic noise and cell specific conductances [73].

It is interesting to note the breadth of computations within the primate brain, but much less is known regarding how they are implemented. Thus, in order to understand the "how" behind these computations, we will focus our attention to three computations that may give rise to some of the other computations: gating, gain control and integration.

Circuit Motifs Dynamic circuit motifs is a concept that attempts to bridge the gap between the structural connections of neurons, their neural signatures (such as their phase and frequency) to the underlying computations that may be encoded by this interaction [93]. These motifs seem to be repeatedly present across species, brain areas and modalities [87], making it an excellent starting point for discovering the "building block" elements of vision.

In this section, we explain the implemented computations by these neurons in terms of the spatial and temporal dimensions of neural representations as these two components are critical to understand and characterize neural signatures as suggested by Turkheimer et al. [87]. In doing so, there is significant explanation based on neural synchronization and oscillations that are not part of the ST model. Here, it is important to note that an explanation of these neural oscillations is provided to give a complete description of the computations in their original context. Since the aim of this thesis is to propose a biological set of elemental operations, the information gained from this section only serves to provide a computational and biological foundation for subsequent sections.

The temporal dimension deals with the role that synchronization and phase have on the computations that may be performed [10, 93]. Here, the synchronization and phase are dependent on the biophysical dynamics of the excitatory and inhibitory cells that make up the circuit motif. The spatial dimension involves the specific connectivity of groups of neurons with the interaction between excitatory and inhibitory neurons playing a fundamental functional role in neural computations [10]. Such connectivity exists between or within cortical and sub-cortical brain regions. Both of these dimensions are dependent on one another and do not exist alone, as such the circuit motifs that will be introduced shortly heavily depend on their interaction.

Feedforward Inhibition The *Feedforward Inhibition* (FFI) circuit motif implements normalization of its input by providing a balance of excitation and inhibition. In addition, FFI is responsible for selecting, integrating and facilitating the propagation of inputs while suppressing or excluding irrelevant information [93]. Furthermore, such a motif has also been shown to perform a multiplication computation of its input-output response curves under certain conditions [93].

The FFI motif is present in a variety of cortical areas and facilitates the transfer of information between the thalamus and cortical layers IV, V, and VI. Between cortical layers within the same cortical column from layers IV/V to LII/III and between cortical areas [93].

Figure 2.6 shows a generic FFI circuit composed of an excitatory pyramidal cell (PC) and inhibitory (I) neurons. The circuit implements filtering of information across the PC neuron. Here, stronger excitatory connections are shown as thicker arrows where the green and red colors indicate excitatory and inhibitory connections respectively.

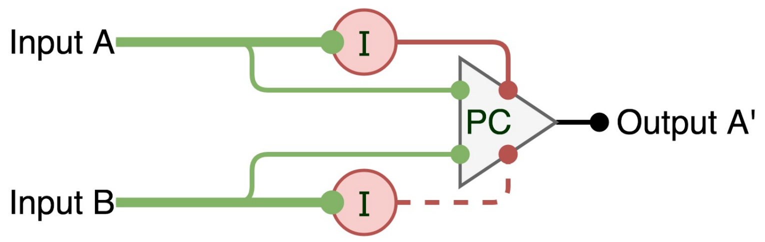


Figure 2.6: A generic feedforward inhibition (FFI) circuit motif consists of strong excitatory connections to inhibitory neurons (red circles labelled “I”) with less excitation to excitatory neurons (triangle labelled “PC”) . Reproduced from [93].

Figure 2.7 shows a FFI circuit with thalamic input projecting to cortical areas in layer IV of the cortical column. The parvalbumin (PV) cells are inhibitory while the PC cells are excitatory. PC cells also project to other cortical layers as discussed previously. In addition to this indirect inhibition, direct inhibition via inhibitory neurotransmitters to excitatory cells is also considered to be an important circuit motif for the projection of information [92].

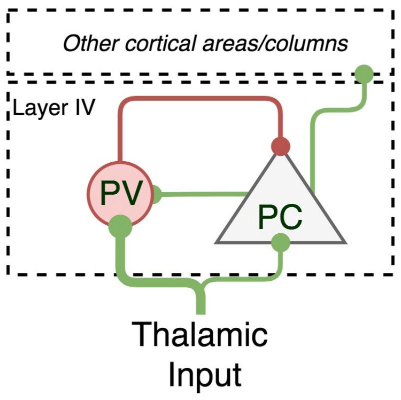


Figure 2.7: A thalamo-cortical feedforward inhibition circuit motif projecting to layer IV of a cortical column. Neurons here project to other cortical areas or columns (see text). Reproduced from [93].

Feedback Inhibition The *Feedback Inhibition* (FBI) circuit motif complements the FFI circuit motif just described by providing an additional gating and gain control mechanism using a well characterized pyramidal interneuron gamma (PING) motif. As with FFI, the PING motif relies on the balance between excitation and inhibition as well as the biophysical properties of the cells that are involved for its computations. The PING motif is involved in the gating and gain control of sensory inputs and is a likely computational candidate for the selective routing of information in the primate brain by enhancing the throughput of attended information and reducing the interference from irrelevant cells [93].

The aforementioned computations of gating, gain control and integration are all dependent on the context or the state of an organism including their motivations and behavioral goals. This context dependent gating has been well studied in hippocampus areas and two circuit motifs involving dendritic inhibition and disinhibition of PCs have been identified for this purpose. The first circuit motif involves dendritic disinhibition and disinhibition to segregate sensory inputs from memory related encoding in area CA1 of the hippocampus. The second circuit involves a top down gating of L5 PCs via an inhibitory gating mechanism that may then be used to affect local processing.

Figure 2.8 shows a generic FBI circuit that implements input selection and gain control. Although this FBI circuit resembles the FFI circuit presented in Figure 2.6, there are two major differences. First, the FBI circuit has excitatory connections projecting from the PC to the inhibitory neurons. Second, the strength of the connections is different, with weak input connections to the inhibitory neurons shown with dotted lines.

Figure 2.9 shows a pyramidal interneuron gamma (PING) circuit motif receiving input from layer IV of the same cortical column and possibly from higher cortical areas.

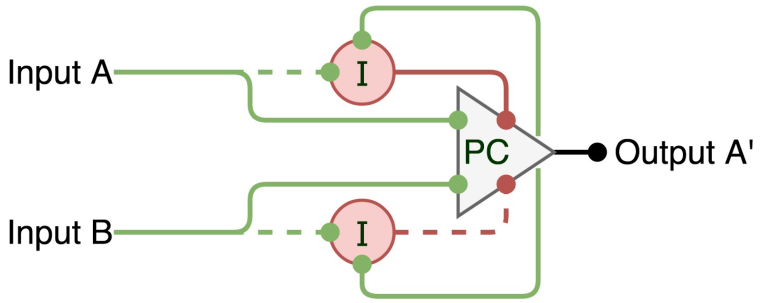


Figure 2.8: A generic feedback inhibition (FBI) circuit motif is almost identical to a FFI circuit except for feedback connections from the PC to inhibitory neurons. Reproduced from [93].

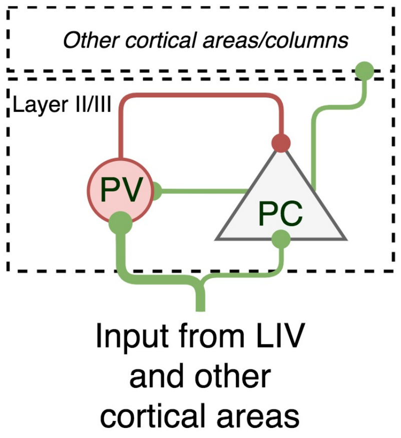

Other cortical areas/columns
Layer II/III
PV
PC
Input from LIV and other cortical areas

Figure 2.9: A pyramidal interneuron gamma (PING) circuit motif provides frequency and phase dependent gating and normalization computations. Reproduced from [93].

Figure 2.10 shows a circuit motif in Layer V of the cortical column receiving input from the thalamus and other layers within the same cortical column. Projections from layer V have been shown to project to other layers within the same column and to lower cortical areas implementing a top down gating mechanism.

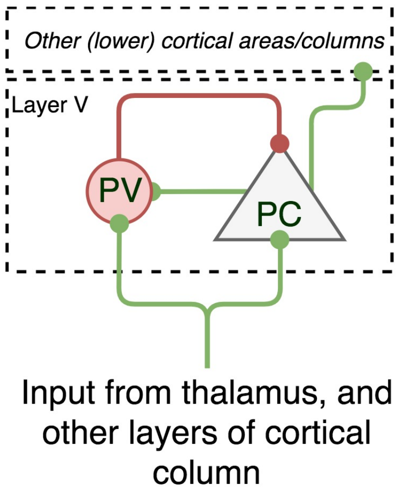

Other (lower) cortical areas/columns
Layer V
PV
PC
Input from thalamus, and other layers of cortical column

Figure 2.10: Feedback inhibition circuit in Layer V of the cortical column projecting to lower cortical areas.

Other Circuit Motifs In [92], several long-range circuit motifs have been proposed that closely resemble the ones mentioned here. The circuits in this paper also show that the propagation of feedforward information from one cortical area to another is complemented with feedforward excitation; with ∼65% excitatory connections and ∼35% of inhibitory connections from the anterior cingulate cortex (ACC) to the lateral prefrontal cortex (lPFC). Hence, a feedforward excitation motif characterized by direct excitation of PCs and a disynaptic disinhibition motif characterized by an excitation of inhibitory interneurons resulting in a downstream inhibition of another interneuron resulting in a net excitatory effect of PCs are two other important circuit motifs.

Studies have also shown that the interaction between two circuit motifs; an intrinsic bursting cell motif with a PING feedback inhibition can lead to a memory maintenance computation by storing the activation in PC cells as a buffer [93]. In addition it has been suggested that this circuit motif may carry information about distinct input streams given the circuit dynamics. Consequently, such a motif may be able to segregate the source of information when parallel input streams are received.

Summary We have introduced several neural computations stemming from neurons and neural circuit motifs that all seem to rely on the biophysical dynamics of neurons (via a balance of excitation and inhibition) and the interaction with other circuit motifs to give rise to a diverse set of computations. Interestingly, the summary provided in Table 2.3 shows that neural circuits can give rise to a small set of repeating computations under different conditions. For each circuit motif, we have provided a reference to the neural circuit.

Computation	**Circuit Motif**	**Figure**	**Description**
Gating	Generic feedforward inhibition	2.6	Information routing and gating of information from thalamus to cortical areas
	Generic feedback inhibition	2.8	Routing of information across multiple cortical and subcortical areas
	Feedback inhibition (PING)	2.9	
	Layer V FBI	2.10	Gating to lower cortical areas via dendritic inhibition
	Dendritic inhibition and disinhibition	Figure 4b in [93]	Input selection found specifically in hippocampus areas
Gain Control	Generic feedforward Inhibition	2.6	Implements normalization (multiplication and division)
	Feedback inhibition (PING)	2.9	Implements normalization (multiplication and division)
	Feedforward Excitation	Box 3, Figure I in [92]	Implements arithmetic computations: addition, subtraction, multiplication, division
Integration	Generic feedforward Inhibition	2.6	Gating and selective integration of information
	Feedback inhibition (PING)	2.9	

Table 2.3: A summary of several elementary computations performed by different circuit motifs. A small set of circuit motifs implement a variety of computations. Each circuit motifs corresponding figure is provided for reference.

2.4 Summary

In this literature review, we have presented the concept of Ullman's Visual Routines as a strategy for completing a visual task using a sequence of operations. From this idea came about Cognitive Programs which are modernized versions of Visual Routines based on what we now know of the neurophysiological literature. Following this, some key aspects of the neurophysiological literature were reviewed including the basic anatomy of a neuron, a cortical column and some important brain areas with their associated computations. This anatomical basis was then used to review the neural correlates of attention at various levels of abstraction followed by an in depth review of the neural circuits that may underlie much of the attentional mechanisms of visual attention. These concepts will be used to develop the foundation of Neural Primitives in the next section.

3 Neural Primitive Specification

3.1 Overview

An atomic operation is defined as "...forming a single irreducible unit or component in a larger system" [76]. We have seen atomic or elemental operations being applied in the domain of making a sandwich [95], as processing steps that are part of some mental algorithms [97], or as atomic operations applied sequentially by Visual Routines to complete a visual task [88].

If the neurophysiological literature suggests that a sequence of operations occur in the primate brain in support of completing a given visual task [47, 61], would it be possible to define the computational counterparts to these biological operations such that a mechanistic version of these operations can be realized? Furthermore, can these computational counterparts then serve as the basis for neurally realistic algorithms? In search to answer this question, we took inspiration from the literature suggesting that the interaction between a very small number of neural circuits can give rise to a variety of computations under different conditions (see Table 2.3). This section aims to propose a mechanistic definition of these biologically inspired computations as a set of elemental operations; the *Neural Primitives*.

To give some context of the usefulness of these NPs within the STAR architecture, recall that the fundamental goal of this thesis is to build a set of algorithms or Cognitive Programs for guiding visual task execution. As part of the construction of these algorithms, NPs play a vital role in this composition. This section is dedicated to the definition of the two major components required for the NP framework; *representations* and *operations*. As such, we focus on the theoretical and mathematical formulation of these components as well as the computations implemented by each of our Neural Primitives. This section does not focus on how NPs may be integrated within the STAR framework. On its own, the NP specification may not seem very useful; instead the flexibility of computations of NPs is realized by their interaction and their algorithmic sequencing and coordination with CPs. However, for now, let us assume that this NP framework serves as the basis for composing useful computations within the Cognitive Programs Memory framework introduced in Section 4.

At its most fundamental level, the NP framework can be modeled as a series of *transformations* that occur on a *base* or *incremental* representation by applying an NP operation. The *Nodes* and *Neural Primitives* are the implementations of the representations and operations, respectively. A brief definition of these terms is provided below.

1. *Representation (Nodes)*: A group of neurons in some matrix of size $A \times B$ encoding some information.

(a) *Base Representation*: Bottom up representation with no operation applied.

(b) *Incremental Representation*: One or more operations have been applied.

2. *Neural Primitives (Operations)*: An operation that defines how a representation will be transformed.

This application of an NP operation is defined as a mapping function $g()$ applied on some representation R at time t to yield some new representation $R' = g(R)$ at time $t+1$ (see Figure 3.1). Note that in this section we focus only on *what* computations these operations implement. *How* these operations are sequenced, coordinated and manipulated is left for Section 4.

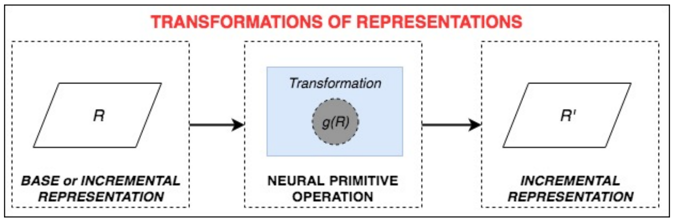

TRANSFORMATIONS OF REPRESENTATIONS
R
BASE or INCREMENTAL REPRESENTATION
Transformation
g(R)
NEURAL PRIMITIVE OPERATION
R'
INCREMENTAL REPRESENTATION

Figure 3.1: A high level representation of the Neural Primitive framework described as a mathematical transformation of the original representation.

If the application of an NP operation is simply defined as a mapping function and we have previously seen that there are several biological operations that may be implemented in the primate brain, then it may be useful to introduce different mapping functions. Each of these functions may implement different neural computational counterparts and have their own neural correlates and uses. For the purpose of the visual tasks that may be implemented within STAR, we propose four NPs: Gating, Bias Control, Bias Feedback and Integration inspired by dynamic circuit motifs [93] and their associated computations as summarized in Table 2.3.

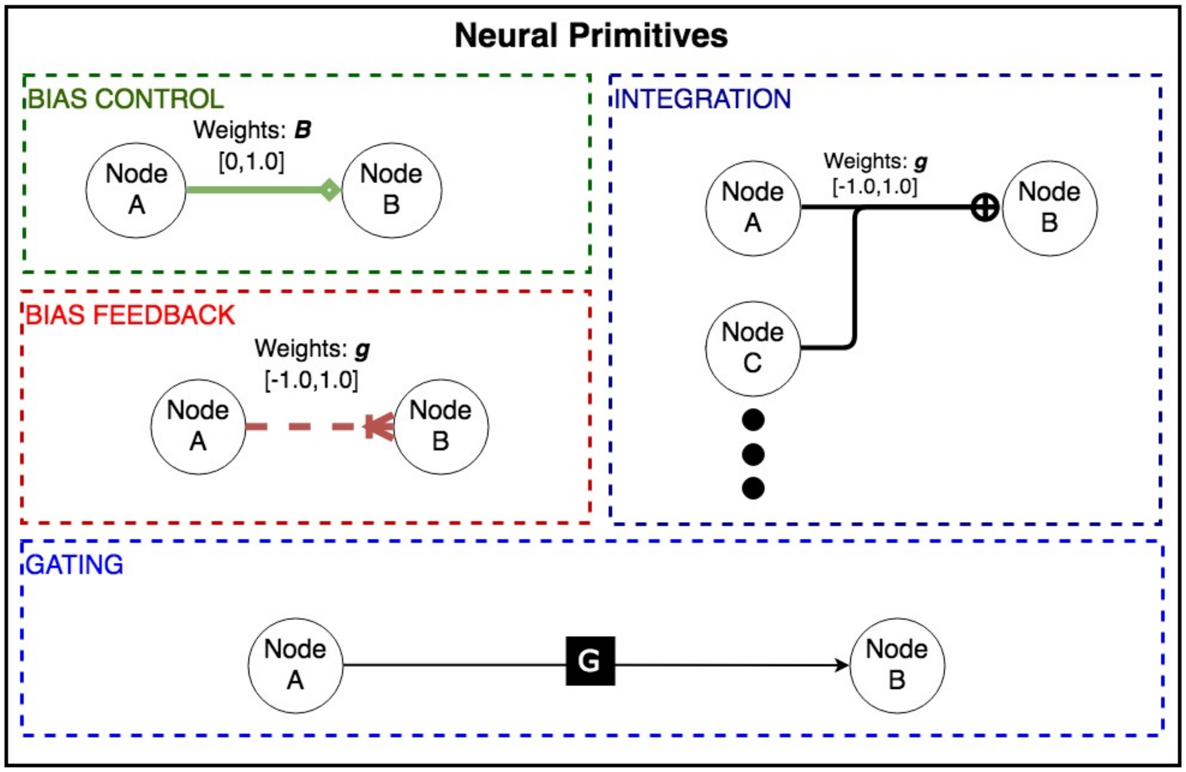


Figure 3.2: The basic elemental operations, referred to as Neural Primitives. Each operation consists of the connection of one or more representations (referred to as a Node) to a destination representation (shown as *Node B* in the figure). A different color and symbol of the arrows is used to indicate each of the Neural Primitives connections. The black square with a capital "G" indicates a gating operation. The associated "Weights" are used to parameterize the Neural Primitives given task demands.

Below we have outlined some of the operations NPs implement either directly or indirectly in conjunction to CP algorithms. For example, the gating and integration computations heavily rely on the CP algorithms to function. In contrast, the bias control and bias feedback operations are able to manipulate representations with minimal algorithmic control.

1. *Gating:* Acts to gate task dependent information between two representations connected together with an NP operation. Action or decision selection may be implemented with the help of CP algorithms.

2. *Bias Control:* Implements a top down multiplicative bias.

3. *Bias Feedback:* Serves to propagate the top down multiplicative bias originating from a Bias Control operation down to the next layer of the VH.

4. *Integration:* Implements evidence accumulation and feedforward information integration with the help of CP algorithms.

In this chapter we describe the theoretical and mathematical notation and formulation of our representations. This is followed by a complete description of the principles, mathematical formulation, neural correlates and response profiles of NP operations.

3.2 Representation

A representation is defined as a group of neurons with some neural activity at a given point in time. Equation 3.1 below is a rewritten version of equation 2.1 for ease of representation of our NPs.

$$r = \frac{1}{\tau} \int [-r + b * S(f)]\, dt \tag{3.1}$$

Where $f = \sum_{k \epsilon \uparrow} \gamma_k g_k e_k(t)$ and S is the function used to compute the neuron's firing rate as a function of time t. The neuron's firing rate S is defined by equation 3.2.

$$S(f) = \frac{Zf^{+}(t)}{\sigma^{\delta} + f^{+}(t)} \tag{3.2}$$

Where Z is the maximum firing rate, f^{+} is the positive half-rectified value of f, δ determines the maximum slope of the function and $\sigma$is a semi-saturation constant that determines the point at which S reaches half of its maximum. See [63] for more details.

The b and g_k terms represent a top down bias and feedforward weights, respectively. The range of values they may take are $0 \leq b \leq 1$ and $-1 \leq g \leq 1$. τ is a decay time constant. These values are manipulated by the NPs and form the core mechanisms by which they operate. However, it is important to note that the lateral weights that were originally present in equation 2.1 have been omitted here since our NPs do not manipulate them. Future work may look at ways of incorporating the lateral inputs into the NP framework.

To extend equation 3.1 to more than one neuron such that we may perform the same computations on a given representation, we define a matrix of neurons R (see equation 3.3) with a size of $A \times B$ as a **NODE** as described in the Overview of this section. Similarly, we define matrices B and F of size A x B for the bias terms and feedforward inputs, respectively.

$$R = \begin{bmatrix} r_{1,1} & \cdots & r_{1,B} \\ \vdots & \ddots & \cdots \\ r_{A,1} & \cdots & r_{A,B} \end{bmatrix} \tag{3.3}$$

To complete the matrix representation of a **NODE,** these matrices may then be combined together as shown in equation 3.4. Each neuron in the matrix obeys the same equation, however the specific b and f parameters may vary between neurons. For example, neurons may receive different feedforward and feedback inputs, thus their f and b terms would differ. Note here that the f and b terms here are part of the B and F matrices, respectively.

$$R(B,F) = \int \left[- \begin{bmatrix} r_{1,1} & ... & r_{1,B} \\ \vdots & \ddots & ... \\ r_{A,1} & ... & r_{A,B} \end{bmatrix} + \begin{bmatrix} b_{1,1} & ... & b_{1,B} \\ \vdots & \ddots & ... \\ b_{A,1} & ... & b_{A,B} \end{bmatrix} * S\left(\begin{bmatrix} f_{1,1} & ... & f_{1,B} \\ \vdots & \ddots & ... \\ f_{A,1} & ... & f_{A,B} \end{bmatrix} \right) \right] \tag{3.4}$$

We will use the notation R' to refer to the transformed representation computed after exactly one time step; specifically R^{t+1}. The NODE will be denoted by R_A where A is the name of the representation. In addition, to refer to the top down bias and the feedforward inputs, we introduce the subscript notations B_A and F_A, respectively. Thus, the short hand notation shown by equation 3.5 outlines the neural activation of NODE A at the current time step with the input parameters B_A and F_A. Such a convention may seem redundant but it is necessary when we begin to talk about multiple representations in the operations section.

$$R_A(B_A, F_A) \tag{3.5}$$

In our work, the NODE can represent a higher cortical area (such as the PFC) providing top down control, visual areas or areas responsible for the maintenance of information. In order to account for the different kinds of nodes, we introduce three subtypes; *Default*, *Memory* and *Context* Nodes. The definition is provided below and the representation symbols are given in Figure 3.3.

1. **Default Node**: In all cases, if there is no mention of the type of node, it is assumed to be a default node. This representation computes its neural activation given by equation 3.4. Each neuron in this representation has its own version of equation 3.1. Thus far we have identified two kinds of default nodes.

    (a) *Visual hierarchy* (VH) nodes where they have also been referred to as sheets in [82].

    (b) *PFC nodes* are responsible for providing as of yet undefined top down control signals.

2. **Memory Node**: This representation works in almost the same way as a default node. The key difference here is that it takes its input at one time point only, after which it maintains its activation even in the absence of neural input. Neurophysiologically, such a neuron may be realized by recurrent excitations as these neurons are found in the prefrontal cortex during memory maintenance. Since integrating a working memory neuron that can

perform this operation is beyond the scope of this thesis, this representation serves to mimic this functionality. Future work may look into incorporating a biologically plausible working memory neuron. Note that this is applicable to the neurons' feedforward input only. A top down multiplicative bias applied to this node may still change its representation. Thus far, we have identified two kinds of memory nodes.

(a) *Working Memory* (WM) Nodes are responsible for predominately maintaining task based stimuli information during visual task execution.

(b) *PFC nodes* may be responsible for maintaining intermediate task state information other than stimuli based information.

3. **Context Node:** This representation is used when we wish to simulate input from a top down source. For example, if task based parameters specify the location of interest, these parameters must be translated as inputs to a representation. To serve this purpose, a context node is used. Our implementation allows for any arbitrary input function to be generated such as $sin(x)$ or $cos(x)$. This node overrides equation 3.5 and instead becomes directly proportional to our input function $R = h(x, y, t)$, where the input function $h()$ is dependent on the spatial position (x, y) and the current time step t.

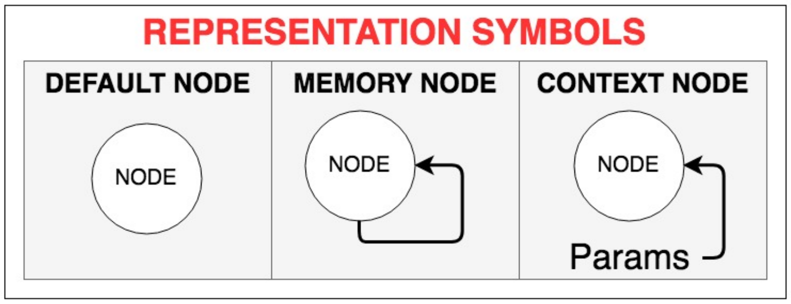


Figure 3.3: Default, memory and context node representation symbols. A memory node is depicted with an arrow pointing to itself indicating the presence of recurrent connections. A context node is represented with *params* as input.

To demonstrate the distinction between a *default* and *memory* node, consider the neural response curves shown in Figure 3.4. Here, a linear input signal is provided for 20ms after which it is removed. Upon removal, the default representation node (in red) begins to decay until its firing rate reaches zero (or baseline), however, a memory representation node (in blue) maintains its activation despite removal of the input.

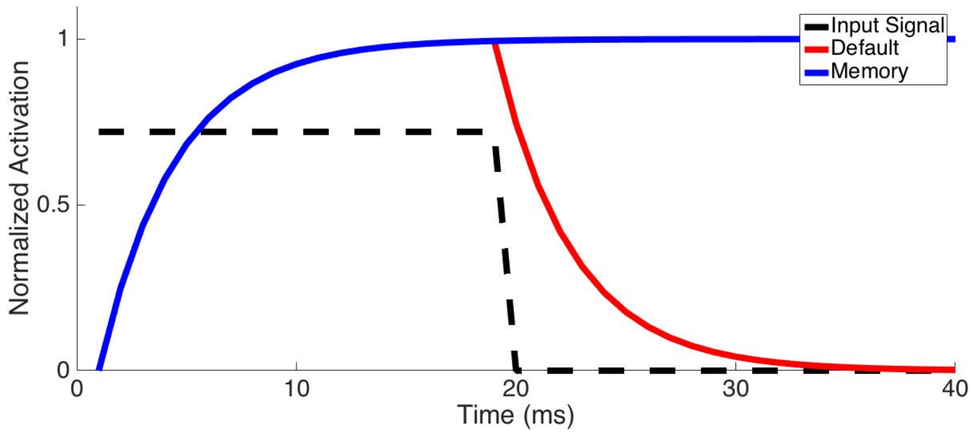


Figure 3.4: Neural activations of a neuron in Default mode (red) and Memory mode (blue) given an input signal (black). When the input signal is removed, the Default node's activation begins to fall while the Memory node maintains its activation.

3.3 Operation

Thus far we have outlined the definition and mathematical formulation of representations. In order to apply a transformation to a representations, we posit that a small set of *elemental operations* - the NPs, can be used for this purpose. These operations are inspired by advances in neuroscience and fundamentally compute a variation of one or more arithmetic computations [73]. Importantly, NP operations form the foundation for our CP algorithms within the CPM framework introduced in a subsequent section. Therefore, in this section, we begin by outlining three *principles* which are true for all NPs. We then use these principles to define in detail the mathematical formulation, neural correlates and response profiles for all four NPs; *Gating*, *Bias Control*, *Bias Feedback* and *Integration*.

The NP framework relies on three key principles that define the mechanisms by which representations are manipulated *(principle 1)*, the constraints on these transformations *(principle 2)* and the neural correlates of connectivity *(principle 3)*. These are detailed below.

Principle 1

An operation works by manipulating a representation through three core mechanisms; the bias (B), the feedforward (F) term and selection of input via gating.

Recall that we have previously defined an NP operation as a transformation of a representation defined by a function $g(x)$ that occurs at one time step (see Figure 3.1). This abstract description indicates the premise of NPs but says nothing of what these functions may be. The four NPs work by manipulating our representations through three core mechanisms; B and F, the bias and feedforward terms, respectively, in conjunction to gating for selection. In this section we will use the notation provided in equation 3.5 to model our NP operations.

Principle 2

A single operation acts to transform one and only one representation (NODE).

Each NP operation acts to transform one and only one representation at any given time step. However, since neurons receive inputs from a variety of different sources at any given time, other NP operations may be transforming the same representation at the same time step. This assertion allows an easier way of understanding what NP operations are acting upon by constraining them to only one representation at a given time.

Principle 3

An NP operation computation can be traced back to its neural correlates of connectivity.

Since our NPs are meant to be the computational counterparts to biological operations, an important question then is "What are the neural correlates of NPs?". To answer this question, we integrated a diverse set of literature indicating neural circuits that are present in different cortical layers of a cortical column [92, 93], the presence of short and long range connections within and across cortical columns [55, 93] and the incoming and outgoing cortical connections within each layer [49, 50]. With respect to neural circuits and circuit motifs, there is evidence that brain networks maximize both the number and the diversity of functional motifs while the structural circuit motifs are small and appear repeatedly [74, 87, 93].

With this information in mind we propose the interaction of excitatory pyramidal cells (PC) and inhibitory parvalbumin cells (PV) in the supragranular (S), feedforward layer 4 (F) and infragranular (I) layers as shown in Figure 3.5. The supragranular and infragranular correspond to layers II/III and layers V/VI of a cortical column respectively. The green and red solid lines indicate excitatory and inhibitory connections respectively while the yellow and green dotted lines indicate feedback connections between different layers of the cortical column. The solid black arrows for the feedforward input and feedback output at layers F and I respectively indicate where this layer may be sending and receiving information from and to other cortical layers. On the right hand side, the *abstract representation* of these neural circuits is represented as a single filled black circle to remove the redundancy from the figure. This abstract representation will be used in subsequent sections when describing the neural correlates for each of the NP operations.

Furthermore, subsequent sections refer to a "connectivity pattern" that indicates the source and destination layers involved in the neural correlates of an NP operation while omitting any intermediate layers the signal may propagate to. This will be written in short hand; for example a F-S connectivity pattern describes connections originating from a feedforward layer 4 (F) and terminating in the supragranular layer (S).

To relate Figure 3.5 back to equation 3.5 of a representation, we assume that the feedforward layer 4 (F) and supragranular layer (S) corresponds to the feedforward F and bias matrices as shown in equation 3.4. Although the infragranular layer (I) does not directly correspond to any of the matrix representations, it serves an important implicit computational role for propagating information within and between cortical layers.

An important aspect of these interactions is a distinction between anatomical and effective connectivity. NPs utilize both short range and long range connections for their functionality. These can be traced back to the layer of the neocortex cortical column to which a neuron sends and receives connections. The *Bias Control*, *Bias Feedback* and *Integration* operations layout the anatomical connectivity [75]. On the other hand, the *Gating* NP can be thought of as a modifier for effective connectivity. That is, the communication of brain networks varies given different cognitive tasks [41].

Finally, although Figure 3.5 shows only one circuit motif, it is important to note that any number of circuit motifs may exist within a cortical layer. This figure was meant to show the *minimum* circuits required to implement our NP computations. Several other circuits have also been discovered and the reader is encouraged to refer to [93] and Table 2.3 for more information.

On their own, the operations are not very useful, but the complex interaction of multiple representations by the Neural Primitives allows for a plethora of useful high level operations. This interaction occurs via the integration of CP algorithms for the purpose of solving visual tasks as we will see in the next section.

3.3.1 Gating

Overview A gating operation is the most intuitive and works by controlling the propagation of information between two Nodes connected together with one of the other NP operations. This operation can be considered as the most important since it controls whether every other operation is active or inactive. Since a NODE can be arbitrarily defined, it need not be an entire visual area or representation. Instead, it may be a subset of neurons within a representation providing rich flexibility in the neurons that may be gated.

Gating may be used to gate the information from the VH to vWM or it may be used to enable priming on the VH. Figure 3.6 shows the computational representation of Gating. The arrow connecting Node A to Node B can represent either a *Bias Control*, *Bias Feedback* or an

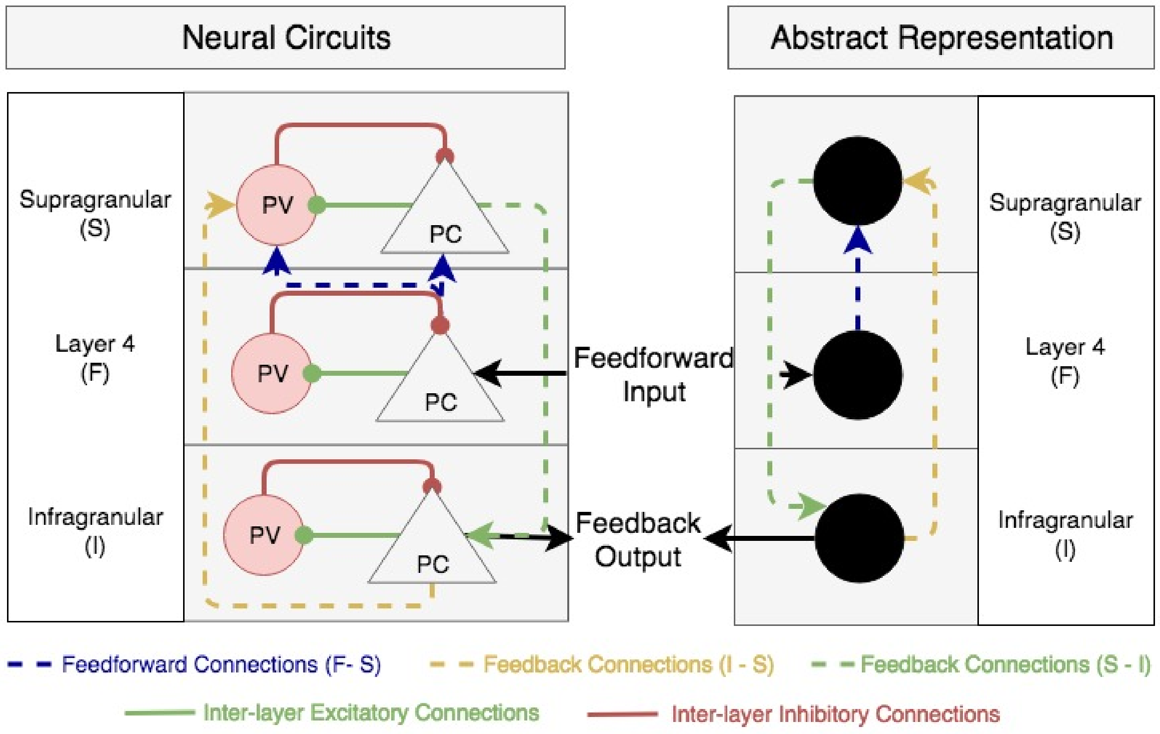

Neural Circuits
Abstract Representation
Supragranular (S)
Layer 4 (F)
Infragranular (I)
PV
PC
Feedforward Input
Feedback Output
Feedforward Connections (F- S)
Feedback Connections (I - S)
Feedback Connections (S - I)
Inter-layer Excitatory Connections
Inter-layer Inhibitory Connections

Figure 3.5: Neural Correlates of Connectivity for the NP operations. The interaction between excitatory pyramidal cells (PC) and inhibitory parvalbumin cells (PV) within and between cortical layers and areas may be responsible for encoding neural computations. The solid black arrows for the feedforward input and feedback output indicate where the layer may be sending and receiving information from other cortical layers.

Integration operation. In the case when no Gating operation is attached to an operation, the default behavior is to always allow the propagation of information between the two Nodes.

During visual task execution, the set of NPs used is defined by the CPs and thus all the NPs that will be used are "fixed" before task execution. However, the NP parameterization can vary on a task by task basis while the propagation of information between Nodes can be dynamically manipulated during a task using the Gating operation. Thus, the information provided by the vTE and vAE is critical to the execution of a CP and to appropriately gate information in a task dependent manner.

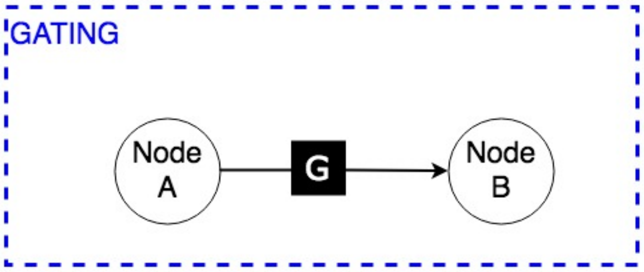


Figure 3.6: A Gating Neural Primitive operation algorithmically controls the flow of information between two Nodes connected together by one of the other NP operations; *Bias Control*, *Bias Feedback* or *Integration*.

All gating occurs in a context dependent manner, that is, it depends on the organisms task state, motivations and goals [92]. Thus, this gating operation does not implement any decision making component. Instead, it is merely the result of a decision given current task demands implemented with our CP algorithms. In other words, CP algorithms may manipulate the gating operation to control the flow of information. For example, if we wish to enable gating of one operation only under certain circumstances (i.e. a neuron's activation exceeds some threshold), this NP can be manipulated by CPs to do so.

Neural Correlates A plethora of literature exists on the role of gating in the primate brain and can be found in [22, 54]. Gating of information is a fundamental computation seen in the primate brain and it has been shown to be implemented via several different mechanisms that may be associated with their neural dynamics [93] or through dendritic inhibition and disinhibition of pyramidal cells (PC) [34].

Formulation The gating operation serves to control the flow of information between any two representations that are connected together with a corresponding NP operation. Given R_A and R_B, as source and destination nodes, respectively, information will only flow from A to B if and only if the corresponding gating unit is enabled.

Response Profile Figure 3.7 shows the effect on a neuron's activation when gating is disabled at $t = 30$ ms (indicated by the dotted line). Disabling the gating function removes the flow of information and hence results in a sharp drop in neural activation. To generate this figure, we connected two Nodes together via an Integration operation with the source node modeled as a context node input.

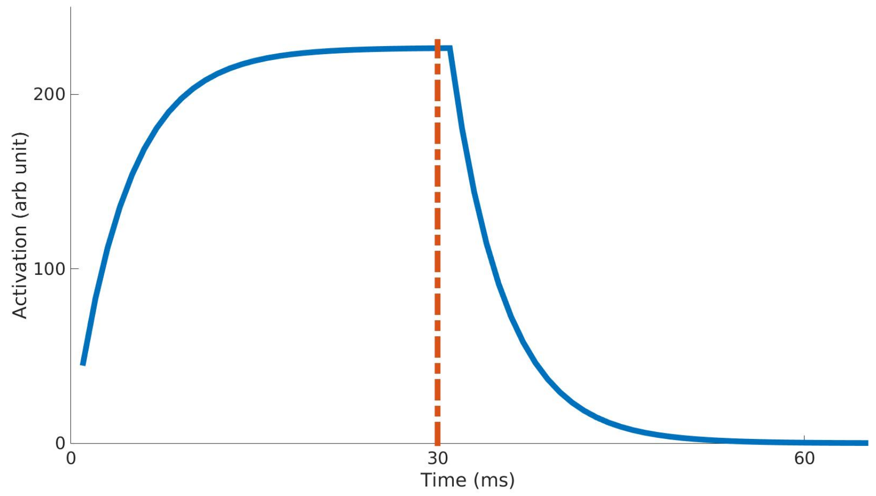


Figure 3.7: Effect of disabling gating between two connected nodes causes a reduction in neural activation. Gating is disabled at $t = 30$ ms indicated by the dotted vertical line at which point the neural activation begins to drop.

3.3.2 Bias Control

Overview The Bias Control (BC) operation implements a normalization (multiplicative or divisive) computation through feedback connections to some destination node. Within the ST model, the BC can be thought as a top down multiplicative bias that is fundamental to some of the suppression, restriction and selection attentional mechanisms; namely task specific biases such as feature of interest, location cues, suppression of task irrelevant information and inhibition of return [82]. As we will see later, this operation is critical to allow for priming the visual hierarchy for specific locations or features.

Figure 3.8 shows the computational representation of Bias Control while Figure 3.9 shows its neural correlates.

Neural Correlates The exact neural mechanism for Bias Control may involve long-range inhibitory connections to excitatory pyramidal cells (PCs). However, given that excitatory projections are more common [92], long range excitatory-inhibitory connections projecting onto inhibitory parvalbumin (PV+) interneurons may indirectly inhibit PCs [93]. In addition, inhibitory SOM+ cells may be a target for feedback inhibition working to directly inhibit PCs

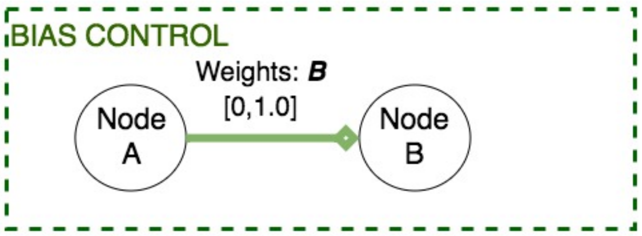


Figure 3.8: Bias Control Neural Primitive.

[93]. Furthermore, an alternative mechanism is that of disynaptic disinhibition which causes a net excitatory effect of PCs by inhibiting inhibitory interneurons.

The bias neurons in the ST model in Layer II/III of the cortical column are the target of the Bias Control (BC) operation. These were originally modeled as an axosomatic process resulting in a decrease in the net post synaptic input [71, 82]. In other words, this process models a multiplicative manipulation of the input-output curve implemented by a set of circuit motifs [73] that is dependent on the interaction between the cell types, the current activation state and the time the input arrives [93].

To represent the neurons and connections that may be involved in this operation, Figure 3.9 shows the descending feedback connections from a higher cortical area (NODE A) to a lower cortical area (NODE B). Bias Control utilizes a feedforward-supragranular (F-S) connectivity pattern as indicated by the orange filled boxes to indicate the source and destination cortical layers the signal originates from and terminates to, respectively. The dotted lines indicate the intermediate cortical layers and neurons the signal propagates from before finally reaching its destination.

The existence of such connections have been found between cortico-striatal and cortico-cortico inter-areal projections [93]. We posit that one mechanism this information may be transmitted is first through a cortico-cortico projection from F-S-I of the same cortical area. Then this information could be projected to I-S to implement a normalization process in lower cortical areas. These anatomical connections may be one of the ways top down information is projected from more frontal to posterior areas [3].

Mathematical Formulation Assume that Node A is connected to Node B using the Bias Control operation, then Node B's bias matrix is directly proportional to the neural activation of Node A as indicated according to equation 3.6. Here, the neural activation R_A is normalized with respect to the maximum firing rate FR_M (see Appendix C for parameters) to ensure the bias term remains within the range $0 < B_B < 1$. ϕ_{AB} is a matrix of real numbers dictating the connectivity weights between the two nodes, taking on values between 0 and 1.

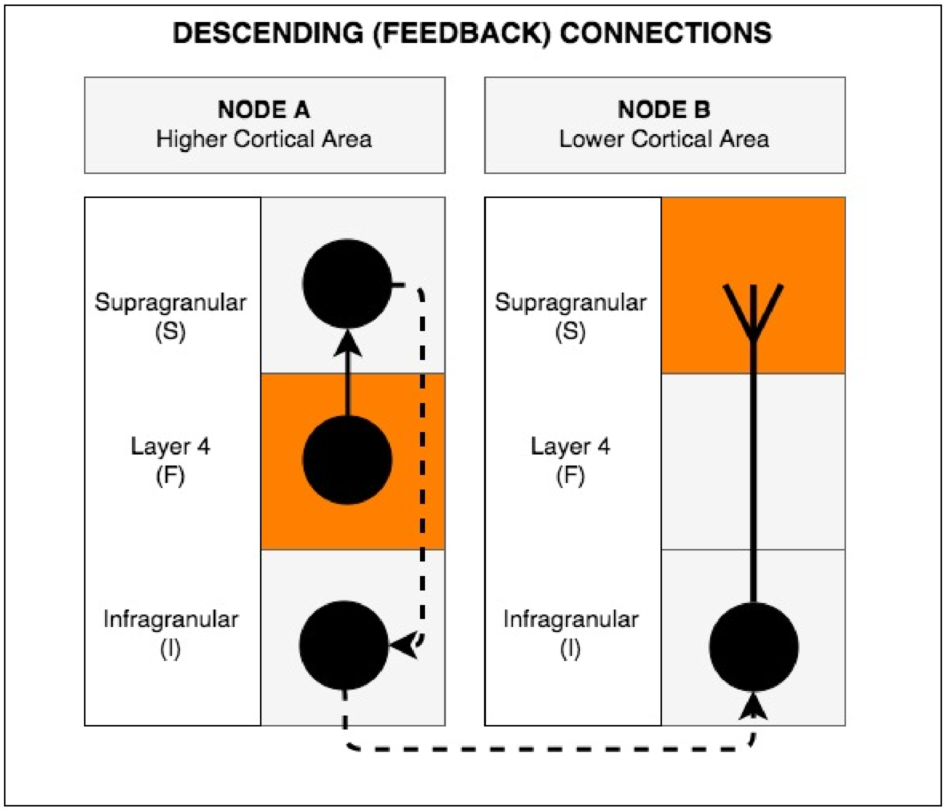


Figure 3.9: Cortical layers and their corresponding neural circuit motifs involved in the Bias Control operation. The diagram is not meant to represent all the neurons that are present in any given cortical layer, only those that are thought to be involved in this operation. Dotted lines indicate connections that are present but do not contribute to this circuit motif. Bias Control utilizes a F-S connectivity pattern.

$$B_B = \phi_{AB} \frac{R_A}{FR_M} \tag{3.6}$$

Plugging this into our matrix representation, equation 3.7 shows how Bias Control transforms the representation. Note that the feedforward inputs F_B are not manipulated by this operation.

$$R_B(B_B, F_B) = R_B(\phi_{AB} \frac{R_A}{FR_M}, F_B) \tag{3.7}$$

The equation above works for the case when the node has only one top down source of attentional modulation. However, what happens when there are multiple sources affecting the bias as is the case with feedforward inputs? Multiple lines of neurophysiological evidence indicate that neurons receive feedback from a multitude of areas, and this must be included in our solution. A simple solution proposed in the formulation of ST was to use the minimum of all feedback bias signals [82], however such a computation does not take into account the interaction of the incoming signals. Instead, we choose to take the mean of all incoming bias signals since it has been suggested that feature and spatial based attentional signals are summed up additively [81]. Other approaches such as taking a weighted average or the median are also viable.

Equation 3.8 shows the generalization for any number of incoming bias signals, where $B_i = \frac{\phi_{iB} R_i}{FR_M}$ and γ_i^B is the gating signal controlled by the Gating operation described earlier. If all the gating signals are disabled, then there is no feedback suppression and the bias term remains at 1.0. Otherwise the mean of all enabled gating signals is taken.

$$B_B = \begin{cases} 1.0 & \sum_i \gamma_i^B = 0 \\ \frac{\sum_i \gamma_i^B B_i}{\sum_i \gamma_i^B} & otherwise \end{cases} \tag{3.8}$$

Response Profile Figure 3.10 shows the bias term (in red) when a normalized input (in blue) is provided by the Bias Control operation before stimulus onset. Here, the normalized input is initially set to 1.0 and at a later time set to 0.5. Before stimulus onset the normalized input is simply set to 1.0 since no top down bias is applied. However, in the case when a top down suppressive signal is provided, the normalized input value drops resulting in a decrease in the bias term as reflected here.

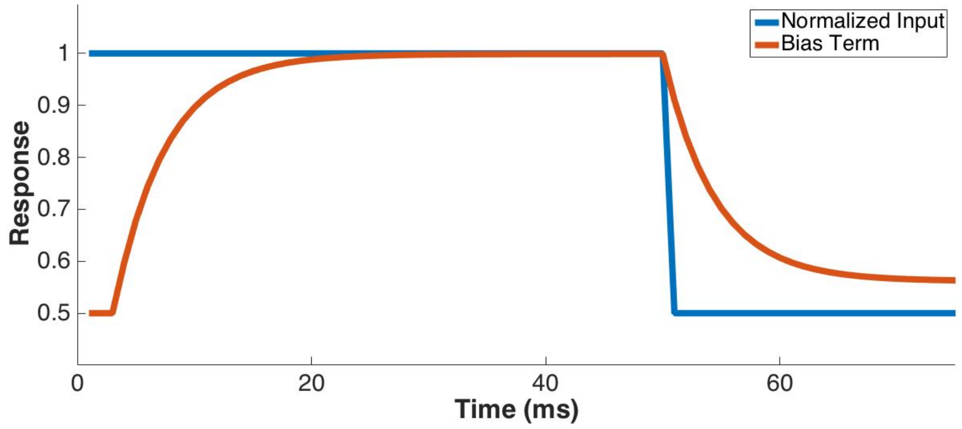


Figure 3.10: Effect of the Bias Control operation on the bias terms neural tuning curve. The normalized input indicates a Node connected to the destination representation through a Bias Control operation. In this figure, the effect of suppressing irrelevant features or locations is shown on the bias terms neural response curve.

3.3.3 Bias Feedback

Overview The Bias Feedback (BF) operation, like the BC operation implements a normalization (multiplicative or divisive) computation through feedback connections to some destination node. The difference however, arises in the specific information being propagated, the neurons involved and the cortical layer the signals project and receive input from. Thus, whereas the

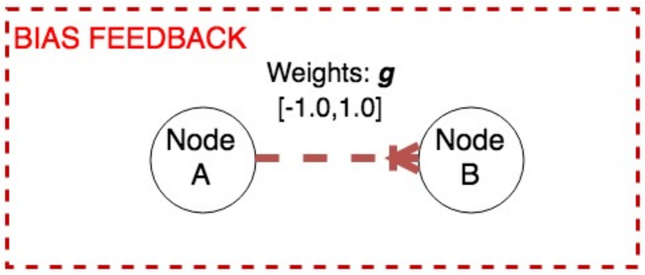


Figure 3.11: Bias Feedback Neural Primitive.

Bias Control operation provided some "top down control", the Bias Feedback operation projects this signal to lower cortical areas while playing a role in local computational processing only.

A clear way to understand this operation is the minimum bias strategy which takes the minimum of all feedback signals and propagates this down to the next layer in the VH as described in [82] and rewritten in equation 3.9 for ease of reference. Note that the bias term can vary as a function of time t. None of the other NP operations allows for the propagation of the bias term B, yet it serves as a fundamental computation for the movement of information. This operation was designed specifically for this purpose, however the minimum bias strategy will not be discussed until Section 4 when we begin talking about this strategy in the context of CP algorithms.

$$B(t) = \underset{\beta \epsilon \downarrow}{\overset{min}{}} \; B_{\beta}(t). \tag{3.9}$$

For example, if a Bias Control operation is applied at the top most layer of the VH that encodes a spatial location, then this encoding must be recomputed for the next layer down in the VH because of the differences in the RF size and eccentricity of the neurons in each layer. How such global information is translated to local information is dictated by this operation.

Figure 3.11 shows the computational representation of Bias Feedback while Figure 3.12 shows its neural correlates.

Neural Correlates Neurons in layer 5 of the cortical column have been shown to have long range projections to other cortical and subcortical structures and may provide a computational mechanism of top down control to other brain areas [49]. Since a variety of neurons exist in layer 5, the nature of the top down control signal may be due to inhibitory (PV^+,SOM) or excitatory neurons. Axons and dendrites of SOM Martonitti cells have been found all across the cortical column but particularly in layer 1. A possible mechanism of suppression of irrelevant features or locations may be that these SOM neurons cause an inhibition of PC neurons within the same cortical layer 5, thus reducing the net feedback excitation to the layers below [49]. In fact, Womelsdorf et al. [93] provide neurophysiological support for such a circuit motif. In the Attentional Routing Circuit model of attention [7], a similar mechanism is proposed for the propagation of a top down control signal.

Figure 3.12 illustrates the cortical layers and cells that may be involved in this operation. The computation begins with bias neurons from the ST equation in the supragranular layer 2/3 (see Figure 2.3) projecting to infragranular layers 1/2 of the same cortical column. This signal is then propagated to feedforward layer 4 of lower cortical areas. As with the Bias Control operation, this operation also incorporates descending feedback connections, but utilizes a S-F connectivity pattern.

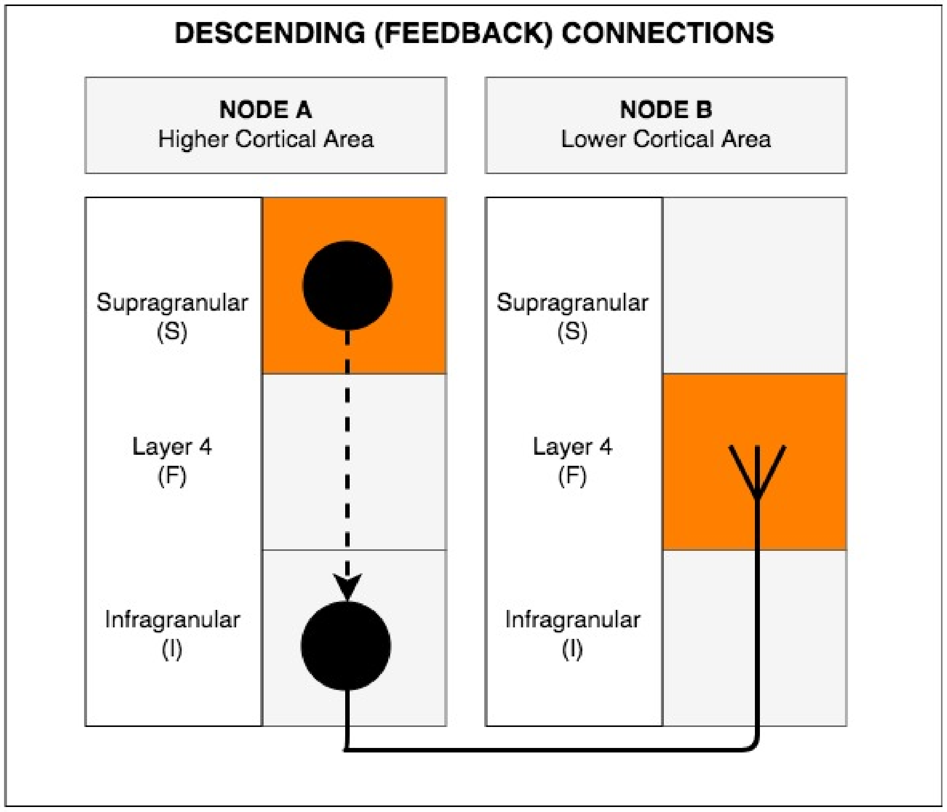


Figure 3.12: Cortical layers and their corresponding neural circuit motifs involved in the Bias Feedback operation. The diagram is not meant to represent all the neurons that are present in any given cortical layer, only those that are thought to be involved in this operation. Dotted lines indicate connections that are present but do not contribute to this circuit motif. Bias Feedback utilizes a S-F connectivity pattern.

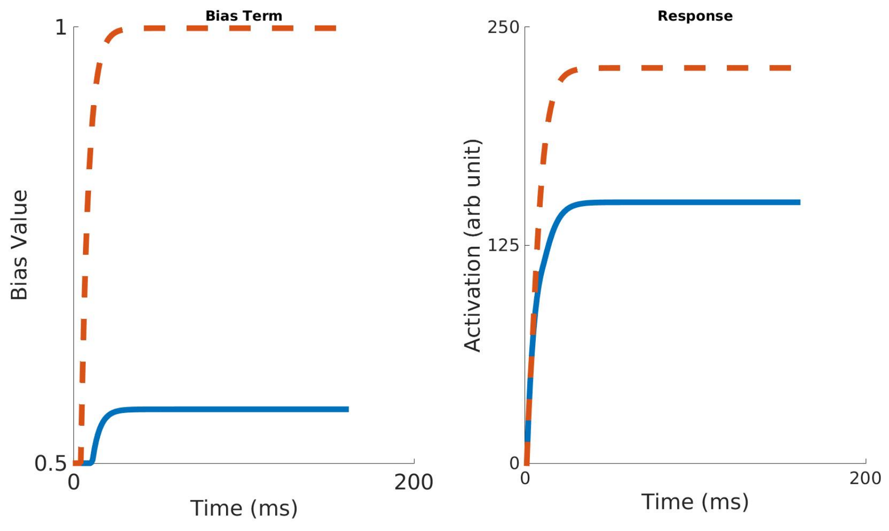
Mathematical Formulation This operation works opposite to that of Bias Control. It allows for the propagation of the Bias term B_A as the feedforward input $F_A = \phi_{AB} FR_M B_A$ to some other node (see equation 3.10, where ϕ_{AB} is the weight matrix connecting Nodes A and B, FR_M is the maximum firing rate that serves as a scaling factor and $0 \leq B_A \leq 1.0$).

$$F_B = \phi_{AB} F R_M B_A \tag{3.10}$$

To manipulate the destination representation, we plug equation 3.10 into equation 3.11.

$$R_B(B_B, F_B) = R_B(B_B, \phi_{AB} F R_M B_A) \tag{3.11}$$

Response Profile Figure 3.13a shows the bias term B_A under two different conditions; $B \sim= 1.0$ (dotted orange line) and $B \sim= 0.56$ (solid blue line) which serve as our source signal that will be propagated from a higher cortical area to a lower one (see Figure 3.12). Figure 3.13b shows the resulting neural response curves R_B color coded for their corresponding input conditions. As expected, a larger bias signal results in a greater neural activation in R_B.



(a) (b)

Figure 3.13: (a) Bias term being propagated B_A, and (b) the resulting neural response curve R_B, plotted under two different conditions.

3.3.4 Integration

Overview An integration operation is the most well known neural computation as it represents the integration of information from a variety of different sources. This information is used in conjunction with CP algorithms for sensory processing, and higher level decision making processes such as evidence accumulation through feedforward excitation and inhibition. Note that a critical distinction here is that the integration operation takes input from a variety of sources whereas the Bias Control and Bias Feedback operations represent a single computational unit between any two Nodes.

Figure 3.14 shows the computational representation of Integration.

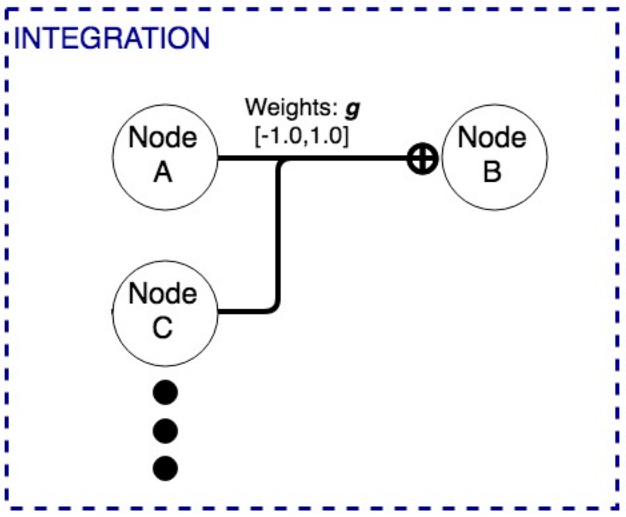


Figure 3.14: Integration Neural Primitive.

Neural Correlates The process of integration is a summation of all the incoming dendritic signals to the cell body [1].

Hence, there may be two possible mechanisms by which information may be integrated to implement a variety of arithmetic computations (addition, subtraction, division and multiplication) [73] as well as more complex higher level decision making processes [8]. First, a feedforward excitation via direction connections to PCs in this layer or feedforward inhibition connections via stronger excitatory connection to inhibitory interneurons [92] . And second, direct inhibition or indirect excitation via disynaptic disinhibition [92] may play a role in this operation.

Figure 3.15 indicates the neurons that may be involved for ascending feedforward connections terminating in feedforward layer 4 of the destination Node B. Note here that the source of the inputs may be direct (through a I-F or S-F connectivity pattern) or indirect since both connections are found in the neurophysiological literature.

Mathematical Formulation This operation allows for the integration of feedforward input and is the most well known configuration of neural connectivity. Equation 3.12 shows the manipulation of the feedforward term F_B while equation 3.13 shows how the neural activations may be manipulated, where R_i represents the integration from the i^{th} representation and ϕ_{iB} is the weight matrix connecting R_i to R_B.

$$F_B = \sum_i \phi_{iB} R_i \tag{3.12}$$

$$R_B(B_B, F_B) = R_B(B_B, \sum_i \phi_{iB} R_i) \tag{3.13}$$

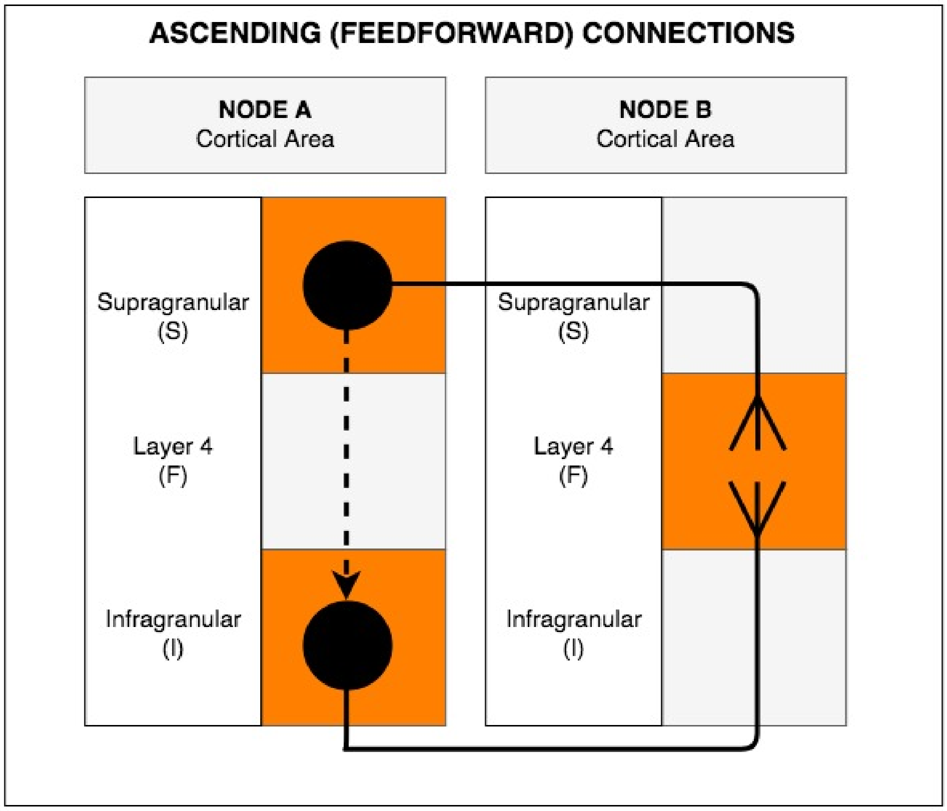


Figure 3.15: Cortical layers and their corresponding neural circuit motifs involved in the Integration operation. The diagram is not meant to represent all the neurons that are present in any given cortical layer, only those that are thought to be involved in this operation. Dotted lines indicate indirect connections that are present. Integration utilizes a S-F and I-F connectivity pattern.

Response Profile The effect of the integration operation is shown in Figure 3.16. The two curves represent the neural activation of R_B when the input signal R_A is set to $R_A = 0.5FR_M$ and $R_A = 1.0FR_M$ denoted by the blue and orange curves, respectively.

3.4 Summary

In this section we have detailed the mathematical formulation and definition of representations and how these may be transformed using NP operations. Two kinds of representations; *default* and *memory* were defined to account for different kinds of neurons found within the primate brain. Four NP operations: *Gating*, *Bias Control*, *Bias Feedback* and *Integration* dictated the specific function computed.

To summarize the material, Table 3.1 shows the three kinds of representations defined; *default*, *memory* and *context* that may have as of yet three kinds of node implementations; *VH*, *PFC* and *WM*.

Table 3.2 shows the derivative computations that may be implemented by our NPs either on

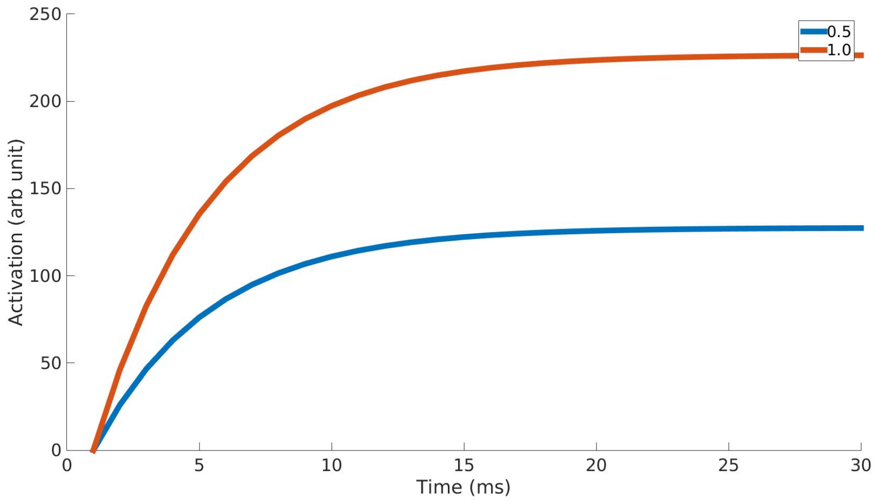


Figure 3.16: Neural activation after applying an Integration operation under two different conditions. When the source input is half $(0.5FR_M)$ and at the maximum $(1.0FR_M)$ firing rate respectively.

	Representation		
Node Implementation	Default	Memory & Context	**Description**
VH	✓	X	Also referred to as VH sheets.
PFC	✓	✓	May be responsible for maintaining task information or application of top down control signals.
WM	X	✓	Predominately maintains task based stimuli information.

Table 3.1: Three kinds of representations (default, memory and context) may consist of one of three types of implemented nodes.

their own or in conjunction to the algorithms implemented by CPs (introduced in Section 4). Such computational counterparts were inspired by biological arithmetic computations [73, 92], circuit motifs (see Table 2.3) [93] and network dynamics [12, 44].

Neural Primitive	**Computations**
Gating	Task dependent information routing, action or decision selection
Bias Control	Gain control, feedback inhibition
Bias Feedback	Feedback inhibition
Integration	Evidence accumulation, sensory processing, feedforward inhibition, feedforward excitation

Table 3.2: The key computations implemented by our Neural Primitives.

Furthermore, the implementation of these computations was defined by a set of mathematical transformations as shown in Table 3.3.

Neural Primitive	**Mathematical Formulation**	**Description**
Bias Control	$R_B(B_B, F_B) = R_B(\phi_{AB}\frac{R_A}{FR_M}, F_B)$	Manipulation of the bias term B_B. Also see equation 3.8 for integration of more than one top down bias signal.
Bias Feedback	$R_B(B_B, F_B) = R_B(B_B, \phi_{AB}FR_M B_A)$	Feedback propagation of the bias term B_A as the feedforward input F_B to R_B.
Integration	$R_B(B_B, F_B) = R_B(B_B, \sum_i \phi_{iB} R_i)$	Integration of feedforward inputs R_i from a variety of different sources.
Gating	▨	Controls the flow of information between two representations.

Table 3.3: A summary of the mathematical formulation of Neural Primitives.

Finally, a fundamental component of this section was to answer the question “what are the neural correlates of Neural Primitives?”. To answer this question, we proposed how groups of neurons within and between cortical layers may interact to give rise to the computations implemented by NPs. Table 3.4 provides a summary of the operations connectivity patterns inspired by anatomical connections found within a cortical column [50]. For the layers we have indicated in brackets whether this layer corresponds to a supragranular (S), feedforward (F) or infragranular (I) layer. Note that this does not necessarily reflect direct cortical-cortical connections between these areas since indirect connections through subcortical areas also exist. Nonetheless, the source and destination areas remain the same. The anatomical connectivity for the gating operation is greyed out since a variety of mechanisms are involved in this process.

Neural Primitive	**Anatomical Connectivity**	
	Source	Destination
Gating	▨	▨
Bias Control	Layer IV (F)	Layer II, III (S)
Bias Feedback	Layer I, II, V (S,I)	Layer IV (F)
Integration	Layer I, II, IV, V (S,I)	Layer IV (F)

Table 3.4: The source and destination connectivity patterns distinguishing each of our Neural Primitives. The letters in brackets S, F and I correspond to the supragranular, feedforward and infragranular layers, respectively, of Figure 3.5.

4 Cognitive Programs Memory Specification

4.1 Overview

Recall that one of the aims of this thesis is to propose a framework for solving visual tasks by applying a sequence of operations on representations until the goals of the task have been met. This was originally envisioned as Visual Routines by Ullman [88].

Cognitive Programs (CPs) - a modernized version of Ullman's Visual Routines [88] are algorithms used to guide visual task execution. These algorithms include computations including but not limited to providing top down control signals, selecting region of interest, dealing with covert and overt fixations, information routing, access to memory, matching to task, priming and coordination of bottom up and top down information [84].

A recent addition of the STAR architecture - the Cognitive Program Compiler (CPC) [32] provides a definition of how CPs can be useful for task execution. The CPC is part of the visual task executive (vTE), and is responsible for translating human readable text such as "a red object will appear" into a task script that consists of a sequence of commands for task execution. Some of these commands may be calls to CPs with parameterization such as $VH.Prime(\{'color' :' red'\})$ while others may contain decision elements for setting up the system and deciding what to do next. The task script generated by the CPC is the starting point for selecting and sequencing the CPs that should be executed for a given visual task.

However, the current STAR architecture still does not provide a standardized way of assembling, storing, retrieving, parameterizing and executing CPs. Because of this, each visual task is hand coded making future development and research difficult and time consuming. Furthermore, the components that can and cannot make up a CP had not been clearly defined. Consider, the four proposed Neural Primitives (NPs) and their computational counterparts inspired by circuit motifs [92, 93] in the previous section. How can we incorporate NP computations into a sequence of operations in CPs such that they may be useful? And how can we design CPs such that they can be reused for a variety of different visual tasks and expedite future research and development into the STAR architecture?

This section introduces the Cognitive Programs Memory (CPM) used to solve the issues of assembling and storing CPs as methods as well as how it interacts with the vTE for CP execution. A *method* acts as a "skeleton" algorithm which cannot be executed without providing task information while a *script* is an executable version of a tuned *method* formed when a task specification is provided.

The CPM is responsible for holding CPs that can be rapidly parameterized and executed with minimal instructions, thus providing ease-of-use for simulating a variety of visual tasks. As the CPM is only a database, it cannot be used to execute a visual task on its own. Instead, it requires the vTE - a fundamental component of STAR, without which task execution would not be possible. The vTE is responsible for retrieving a method from the CPM, set the parameters given task demands to form a script and subsequently execute them. The stored methods in the CPM are algorithms that specify what is required for task execution and are composed of a sequence of commands. Once a CP is finished with its execution, the result is returned to the vTE which then decides on what to do next. A detailed description of how methods are assembled and stored by the *Method Elements* (CPM-ME) and *Methods Database* (CPM-MD), respectively is presented in Section 4.2.

While Section 4.2 deals with how any method is assembled and stored, Section 4.3 introduces eight implemented methods (see Table 4.1) that are useful for task execution. Each method may also act to utilize NP computations for manipulating representations contained in one or more modules of STAR while others may not need to manipulate them at all. As such, the introduction of each method is accompanied by a thorough discussion of the involved *Method Elements*, representations (or Nodes), NP operations and their neural correlates. Future work may look to expand the possible set of methods and the ones listed here are only a starting point. Finally, after thoroughly describing each of the methods, we then explain how a stored method may be parameterized, retrieved and executed by the vTE.

Recognition Methods	**Fixation Control**
VH.Localization	FC.setFHMBias
VH.Prime	
VH.Detection	
VH.Recognition	
VH.Categorization	
VH.Identification	
VH.Identification (within category)	

Table 4.1: Methods are composed of *Method Elements* in the CPM-ME, stored in the *Methods Database* (CPM-MD) and divided based on the STAR module they operate on.

4.2 Framework

Before we delve into the components of the CPM, it is important to step back and look at the CPMs interaction with the rest of STAR as shown in Figure 4.1. Doing this, allows us to visualize the steps required to complete a visual task.

The figure is incomplete and by no means represents all of the components that are present in each of the STAR module as described in Section 2 and [84]. For example, the arrows indicating the flow of information as shown in Figure 2.2 between each STAR module have been omitted in Figure 4.1 for clarity. To make the material more digestible, the figure serves as a bare-bone STAR framework where we can add in additional components such as representations and their corresponding NP operations as required by each method that will be introduced in the subsequent section.

For visual task execution to occur, methods must first be assembled using *Method Elements* (CPM-ME). Thus far, assembled methods are hand crafted but future work may look at automatically learning new CPs given the current task demands. Assembled methods are then stored into a *Methods Database* (CPM-MD).

Just before visual task execution, the vTE fetches the relevant methods from the CPM and sets the appropriate parameters to all other STAR modules. The vTE then sequences their execution and makes deterministic decisions on whether to terminate the task or execute another CP depending on the current task state. The script constructor, executor and monitor in the vTE are represented as a single block that takes in task specification as input. Further detail of the components may be found in Section 2 and [32].

The vAE receives control signal specification from the vTE dependent on the specific CP to be executed and these signals are propagated to other STAR modules - VH, FC, vWM and tWM using the control signal generator. The vAE may also access the active script notepad in the tWM to read currently set control signals in order to assess the current state of the system. This information may be related back to the vTE to make decisions such as whether the task is complete. Some of the possible control signals are indicated by arrows connecting the control signal generator to the VH. Refer to [84] for further discussion of the involved control signals.

The VH receives a feedforward retinal signal and processes input through layers V1, V2, V4 and IT. Here, each layer is composed of N sheets, each representing a selectivity to a certain feature such as color or orientation. The parameters used for the experiments are found in Appendix C.

Now that a high level understanding of how a task may be executed is provided, it is important to look at how methods may be assembled and stored in the CPM in more detail. Figure 4.2 shows that the CPM-ME is itself composed of several subelements: NP operations as discussed in Section 3 and Base Methods that are of themselves composed of several subelements. The idea of assembling methods is that Base Methods can be sequenced together to form more complex ones. Once a desired configuration is achieved, they can then be stored (*Stored methods*) in the CPM-MD. A detailed discussion of the CPM-ME and CPM-MD is provided in Sections 4.2.1, and 4.2.2, respectively.

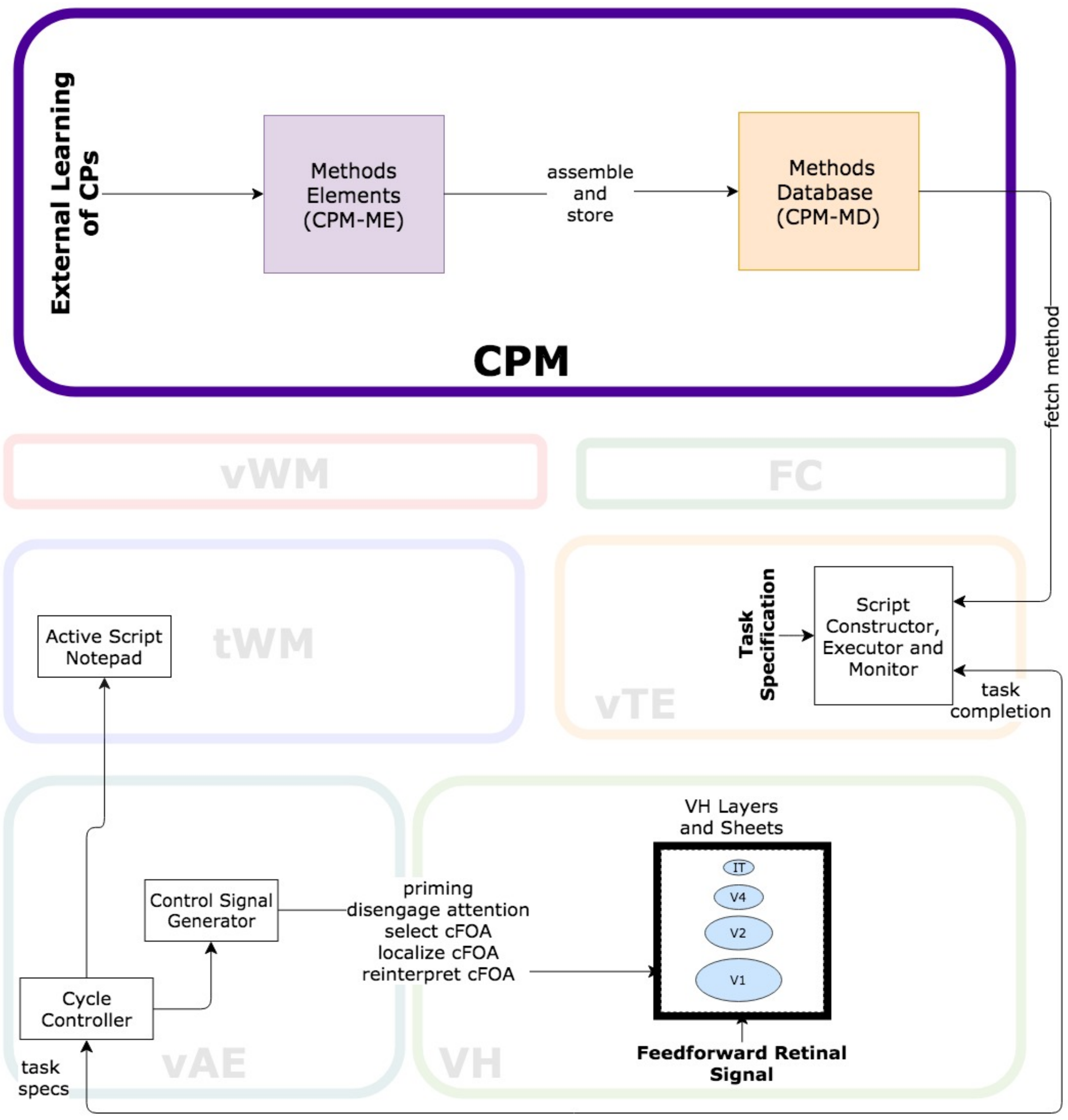


Figure 4.1: A bare-bone STAR architecture showing a subset of the components involved. Arrows indicating the flow of information between the various modules have been omitted as well as several other components for clarity. Here, the Cognitive Programs Memory (CPM) is responsible for the creation of new CPs using a set of Method Elements (CPM-ME). CPM-MEs are then assembled and then stored into the Methods Database (CPM-MD). Stored methods are accessible to the vTE for retrieval, parameterization and execution. Additional components in each STAR module will be added as each CP is introduced.

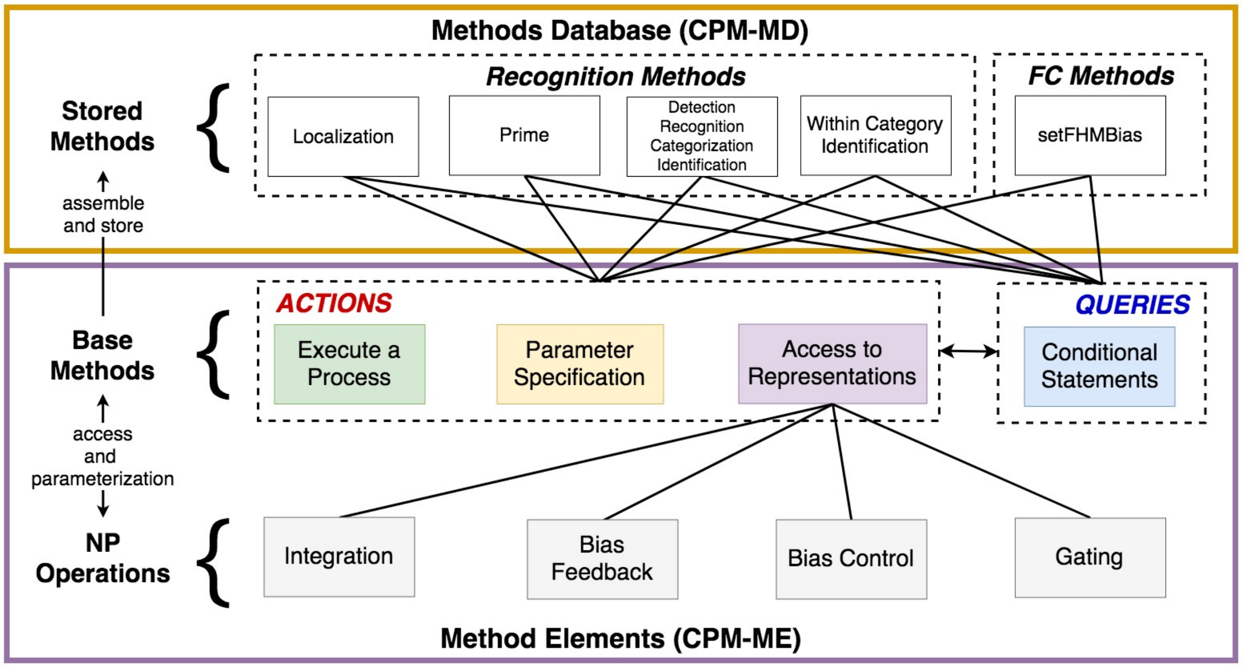


Figure 4.2: The Cognitive Programs Memory (CPM) is composed of the Method Elements (CPM-ME) and the Methods Database (CPM-MD). The CPM-ME uses low level NP operations in conjunction to a set of Base Methods to compose useful methods for task execution. Such methods are then stored into the CPM-MD. The arrow between Actions and Queries indicate that information captured from executing a process, parameter specifications and access to representations can be used for conditional statements. This Figure is identical to Figure 1.1 introduced in Section 1.

4.2.1 Method Elements

Just as Ullman proposed a theory that visual tasks can be solved by reducing the required processes down to a set of atomic operations that may be parameterized for a given task, the *Method Elements* (CPM-ME) as part of the CPM serves this purpose.

This component provides a set of commands - Base Methods, that can be combined in a number of different ways to form more complex methods that may be utilized for visual task execution. In Figure 4.2, the arrows between two categories of Base Methods - Actions and Queries indicate that information captured from the execution of a process, parameter specifications and access to representations can be used for making future decisions. The arrows connecting the NP operations to the “Access to Representations” indicate that Neural Primitives can be utilized by Base Methods to perform computations on representations, as discussed in Section 3. Notice that the total breadth of commands that we require is beyond those computed by the NP operations. For example, several of the control signals such as *apply suppressive surround* or *disengage attention* are not part of the NP operations. Future work may look to propose additional computational counterparts of biological operations that would extend the current set of NP operations.

Below, we describe all of the Base Methods that are divided into two categories - Actions and Queries. For each category, there are several implemented Base Method function calls with their corresponding parameters. All of the methods in the CPM-MD that are discussed later will be constructed from this set of commands.

1. *Base Methods*: Set of commands that can be used to construct other methods, divided into two categories.
   (a) *Actions*: Actions help to bring the system closer to task completion in a number of different ways. Three kinds of actions are defined: Access to representations, parameter specification and execute a process.
   (b) *Queries*: Queries are encoded as conditional statements. They can be used to read the current state of the system and the result provides information on what the next action should be.
2. *NP Operations*: Operations act to transform representations (see Section 3) and are utilized by the Base Methods.

Actions Access to representations includes reading the current neural activations of a Node or enabling and disabling the NP Gating operation connecting two Nodes. Reading the current neural activations may be useful for making decisions on what should be done next (such as task termination), while manipulating the Gating operation allows for the movement of information (information routing). Enabling or disabling the Gating operation is done depending on current task demands at an instance in time. Additionally, writing to a Node is used when setting the input parameters, as is the case for context Nodes (see Section 3.2). Table 4.2 shows three implemented commands for this purpose.

Base Method	**Description**
Gate <source node> to <destination node>	Enables or disables NP gating between two Nodes connected by one of three NP operations (Integration, Bias Feedback and Bias Control).
Read <node>	Reads the current Node state.
Write <node>	Writes to the Node state.

Table 4.2: Access to representations is composed of three commands. <> brackets indicate the allowed parameterization for the functions.

Parameter specification, as shown in Table 4.3, is useful for initializing various settings before subsequent CP execution. For example, if we wish to prime the VH using an attentional sample

(AS) stored in the tWM, then the appropriate input for the tWM context nodes must first be set. Furthermore, parameter specification is also useful for specifying the decision criteria for determining if a match operation is successful.

Base Method	**Description**
Load <AS> to <tWM,vWM>	Loads the AS to either the tWM or vWM by utilizing the "Write <node>" command in *Access to Representations.*
Set <Decision Criteria, Mem Consolidation> Params	Sets the parameters for detection and the memory consolidation signal.
Set FHM [task bias] for <location, weight>	Sets the parameters for FHM bias.

Table 4.3: Parameter specification commands. <> brackets indicate the possible parameterization for the functions, each separated by a comma.

Finally, execution of a process refers to either selection of another CP for execution or another command that should be executed. For example, after a single feedforward pass through the VH, a winner needs to be selected at the top of the VH using the $\theta - WTA$ algorithm. Such an action can be used to wait until a winner is selected or perform another action if no winner is found. See Table 4.4 for a summary of commands.

Queries Queries are encoded as conditional statements and the result is used for deciding what the next action should be. The current state of the system can be read by reading the current neural activations of representations or by reading the current task and control signal parameters stored in the active script notepad. Some of the useful queries we use are listed in Table 4.5.

4.2.2 Methods Database

Thus far we have highlighted a set of Base Methods in the CPM-ME, some of which utilize NP operations while others do not. Now, the *Method Elements* must be combined in meaningful and useful ways to give rise to computations involved in a variety of visual tasks. Method element sequencing and parameterization, independent of task specification results in the assembly and storage of methods in the *Methods Database* (CPM-MD). The Methods Database (CPM-MD) contains the implemented methods listed in Table 4.1 and a detailed description of each one is provided in Section 4.3. This list is not meant to be exhaustive, but instead they show a proof of concept of the required CPs for executing several visual tasks.

Methods from the CPM-MD may now be fetched by the vTE which can then set the ap-

Base Method	**Description**
Feedforward (FF) retinal Signal	Enables the propagation of the retinal signal through the VH.
Select central Focus of Attention (cFOA)	Attempts to make a selection at the specified layer using the $\theta - WTA$ algorithm. See Literature Review Section 2.2.
Apply suppressive surround (SS)	See Literature Review Section 2.2.
Disengage attention	Disengages any attentional mechanisms in place.
Do Location Inhibition of Return (L-IOR)	Engages the L-IOR mechanism. See Section 4.3.8.
Select <Layer Name, NEXT> Layer	Selects the specified layer.
Select <Method Name> Method	Allows for the selection of another method to be passed off to the vTE for execution.
Build <AS> from <params>	Builds a task specified AS that may be used for other processes (for example, by "Load <AS> to <tWM,vWM>" in *Parameter Specification*).
Reset <vWM,tWM>	Resets neural activations of the vWM or tWM.
Compute Decision	Computes the result of VH.Detection, VH.Recognition, VH.Categorization, VH.Identification using equations presented later.

Table 4.4: Commands for executing a process. <> brackets indicate the possible parameterization for the functions, each separated by a comma.

Base Method
Is the winner found?
Are there more layers?
Is there AS for this layer?
Start Layer Specified?
Store AS?
Decision Criterion Met?

Table 4.5: Queries are encoded as conditional statements and the result is used for deciding the next state of the system.

propriate parameters to make executable scripts provided some task specification. For example, one of the methods shown in Table 4.1 - *VH.Prime* is used to provide stimulus expectations before it is presented to the system. If the system is told that "a red object will appear", then this parameterization must be translated into a task specification. In the next section, we will explain how the CPM-ME constituents are used to assemble methods and how they are effected by task specification.

4.3 Cognitive Programs Memory Methods

4.3.1 Overview

In this section we introduce each of the methods in the CPM-MD, listed in Table 4.1. For each of the methods, two key diagrams will be presented.

The first diagram will show a flowchart-like algorithm of the *Method Elements* used to compose a method, with arrows to indicate the flow of execution. As part of this, each of the methods also require access to one or more representations.

The second diagram shows the representations as Nodes and the corresponding NP operations in the STAR modules. In order to reduce redundancy and improve clarity of our figures, connections will be shown in the compressed view as indicated in Figure 4.3. In this example, a Bias Control operation from each layer and sheet in the tWM makes a connection to the corresponding layer and sheet in the VH. This imposes a constraint, namely that the number of Nodes in both the VH and tWM are identical. A gating node is optionally shown between an NP and its default value is set to *false* unless otherwise specified.

4.3.2 VH.Localization

Overview During a single feedforward pass, only the information at the top most layer is available, and sometimes this may not be enough to correctly detect, discriminate or categorize a stimulus. In such a case, the localization process may be invoked to iteratively traverse down the VH while applying a suppressive surround mechanism at each layer to improve the signal-to-noise (SNR) ratio. The VH may then be "reinterpreted" in an attempt to reclassify the stimulus [85]. For a detailed description of the localization process see [85].

The localization process also results in the formation of the attentional sample (AS). The result of the *VH.Localization* process is a reference to a partial or a fully defined AS. A partial AS is formed when the localization process terminates early due to an interruption by another process or by task specification. In this case, a full traversal down the VH has not occurred and only a subset of the information is captured. This AS can be used to parameterize the input of

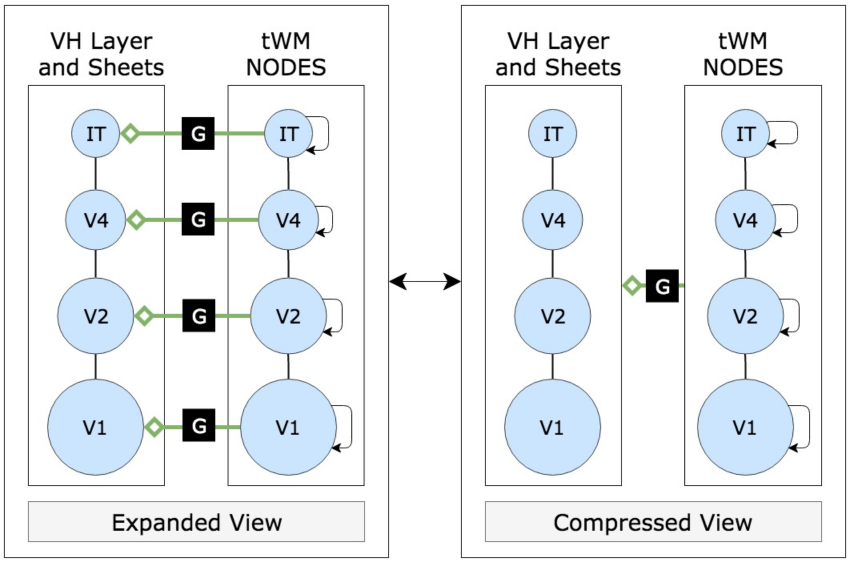


Figure 4.3: For clarity of subsequent diagrams showing the connection of Nodes between different STAR modules, the expanded view will be represented using the compressed view convention. See text for further description.

other CPs such as for priming the VH, recognizing the class of a target or biasing the fixation history map (FHM).

As with any other method, *VH.Localization* contains *Method Elements* that operate on components within STAR. Figure 4.4 shows an integration operation allowing information to be routed from the VH to the vWM to build an AS during the localization process. Although the active script notepad is not used in this method, it is included here to show that the state of the vWM can be read using this module. Furthermore, several of the control signals used are also indicated.

Method Elements - Algorithm Figure 4.5 shows the sequence of actions and decision elements that compose this method and act on the subset of components highlighted in Figure 4.4.

This method begins with first reseting any contents that may be present in the vWM (*Reset vWM*) followed by a feedforward pass (*Feedforward Retinal Signal*) of the VH. A winner is then selected at the top of the visual hierarchy (*Select cFOA*) and the $\theta - WTA$ decision process is invoked. If a winner is found (*Is the winner found?*), the resulting cFOA is gated to the corresponding vWM Nodes using an integration operation (*Gate (Selected) VH Layer to vWM*). The suppressive surround (SS) mechanism is gated algorithmically (*Apply Suppressive Surround*) and further discussion of this is beyond the scope of this thesis but see [82].

Next, if there are more layers (*Are there more layers?*) the process is repeated in the layer down (*Select NEXT layer*) and all the neurons that do not contribute to the winner in

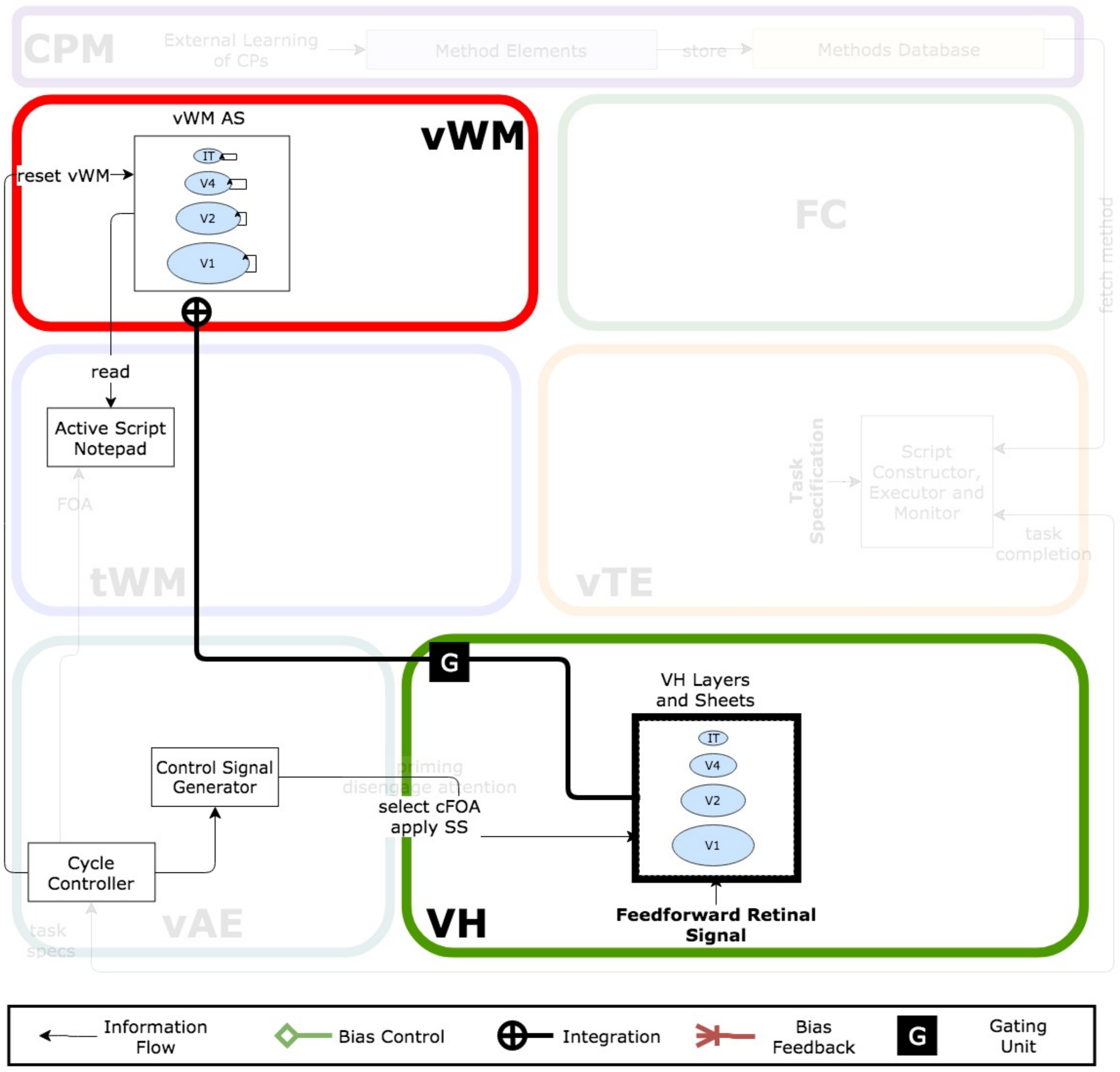


Figure 4.4: During VH.Localization, an AS is built and stored in the vWM using an integration operation from the VH. The resulting AS may be read by the Active Script Notepad for subsequent operations. The vWM contains Memory Nodes indicated by arrows pointing to itself.

the higher layer are pruned, in effect improving the SNR from the figure and surround. This process continues down the visual hierarchy and the resulting AS can then be stored into the vWM for later use.

In some cases a full traversal down the VH may not occur given either task demands or an interruption by some other process. For example, task demands may specify that it is sufficient to know which quadrant the stimuli appeared in or incoming visual stimuli may prevent localization to occur down to the lowest layers of the VH. In such cases, we end up with a partial AS which is returned to the vTE that then decides the next course of action.

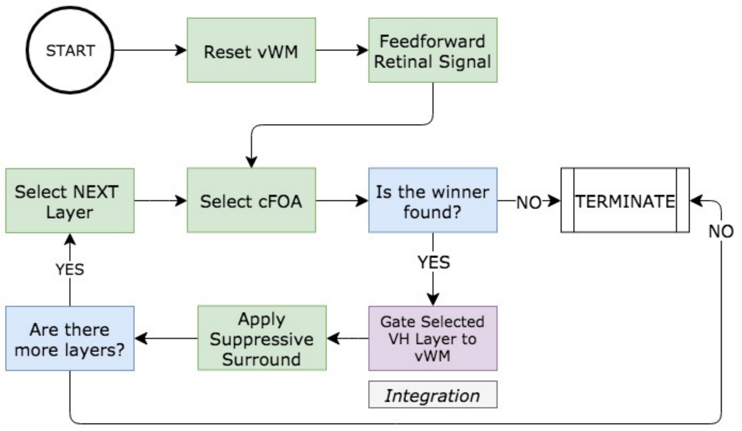


Figure 4.5: The VH.Localization algorithm shows the sequence of actions and queries that are required for CP execution. Each box is color coded with access to representations (purple), parameter specification (yellow), execute a process (green) and conditional statement decision (blue).

Neural Correlates The localization process requires an integration of a variety of neural mechanisms and is beyond the scope of this thesis. Instead, neurophysiological support for localization, the suppressive surround mechanism amongst others can be found in [85]. The resulting attentional sample produced by this CP may reside in a "blackboard" with neural correlates previously proposed to be in the pulvinar [31] but more recent work suggests that the vWM with neural correlates in the PFC may be necessary to store a full attentional sample [70].

Localization Example Figure 4.6 a shows the incoming visual stimulus to the VH which then passes through layer V1 of the VH (Figure 4.6 b). Figure 4.6 b shows the AS representation in the vWM after the contents of VH layer V1 have been gated. However, since the AS is a hierarchical network of winning units, we can choose to keep only the information contained in these winning units. In order to do this, the AS may be transformed to represent location information only as shown in Figure 4.6 c. Here, the white area represents the subset of winning units all of which are set to 1.0. The surrounding units, shown in grey are suppressed.

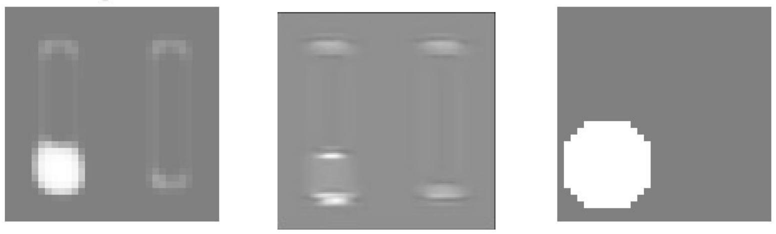

(a) (b) (c)

Figure 4.6: Copying an attentional sample (AS) from VH to vWM. (a) Incoming visual stimulus to the VH. (b) The representation of an AS from layer V1 in the VH stored in the vWM and (c) The transformed attentional sample for representing spatial location only.

4.3.3 VH.Prime with AS

Overview Priming increases the selectivity of relevant features while suppressing irrelevant ones and in order to be effective, it must be applied 300 - 100 ms before stimulus onset [48]. Specifically, priming can be used to provide useful information to the subject as to where they should look or the kinds of features and objects they should be searching for. Such information can result in a reduction in reaction times and error rates [56] as well as a modulation in neural tuning curves [39, 40].

One way to prime the VH is to present a cue to the system to be processed, build an attentional sample and use this as a basis for priming as discussed in [84]. This method describes how an attentional sample built from *VH.Localization* (see Section 4.3.2) can be used for priming the VH for spatial, feature or object based information.

Priming with an AS requires the Nodes in the tWM and VH connected together with a Bias Control operation as shown in Figure 4.7. The tWM is represented as a context node and takes as input an AS for parameterization. The corresponding control signals required for priming are also indicated by the Control signal Generator.

Method Elements - Algorithm To prime the VH using an Attentional Sample (AS), one must already have been created by the VH.Localization process.

Depending on what we wish to prime for, we can have an AS represent three different kinds of information as listed below. For more information about these different AS representations, please see Appendix C.

1. *Spatial AS*: The captured AS only contains information about the location of the object.
2. *Feature AS*: The captured AS only contains information about a feature of the object (i.e. “the object is red”).
3. *Object AS*: The captured AS contains both location and feature information about the

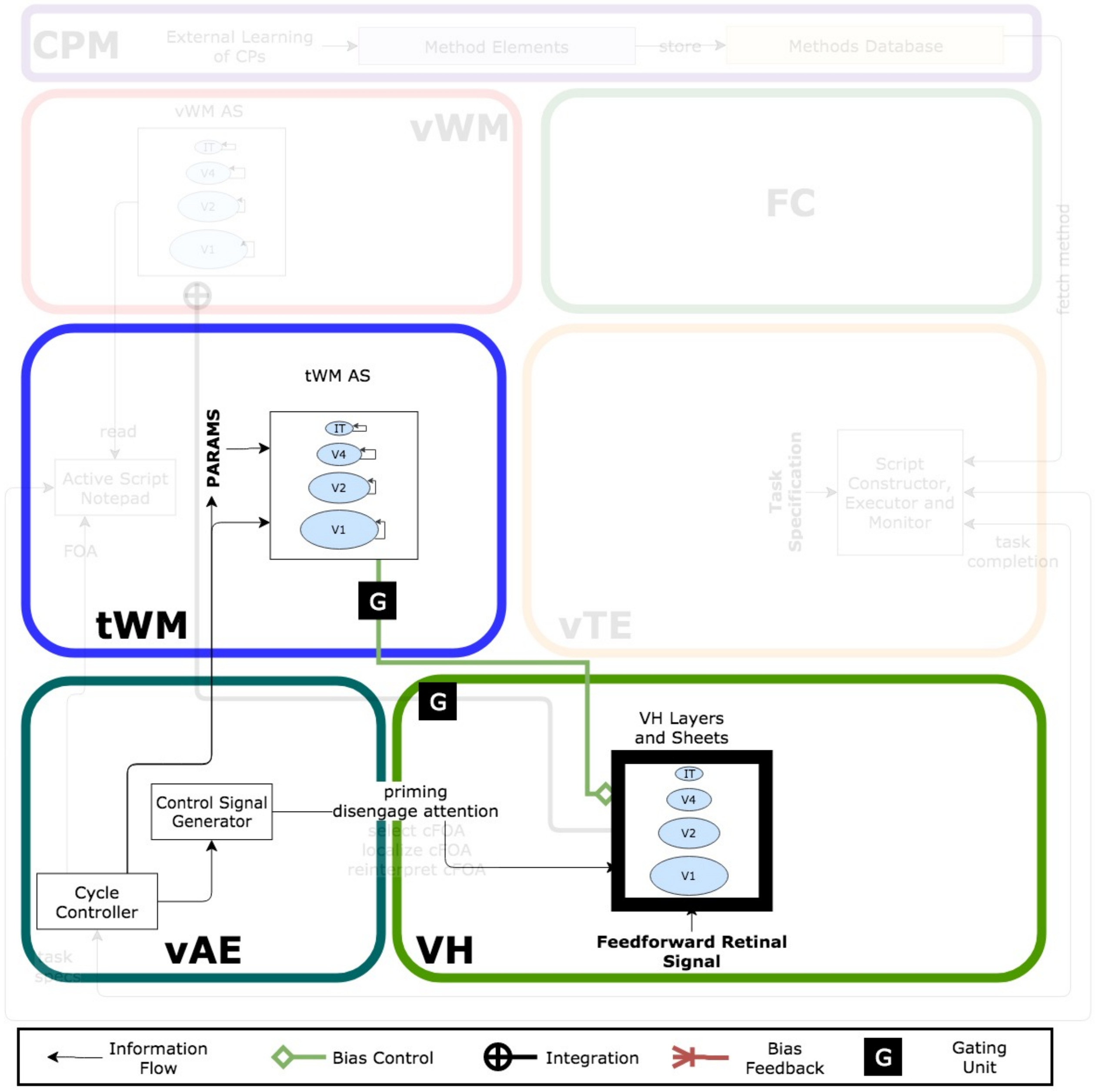


Figure 4.7: Priming with an attentional sample (AS) requires Memory Nodes in the tWM with context node input provided by the vAE. The tWM primes the VH with an AS using the Bias Control operation.

object.

As an example, Figure 4.8 shows an AS containing only location information on three layers of the VH: IT, $V4$, $V2$ with the corresponding sizes indicated in brackets. For this example, it is assumed that the feature information is irrelevant, hence the AS is identical for all sheets within a given layer.

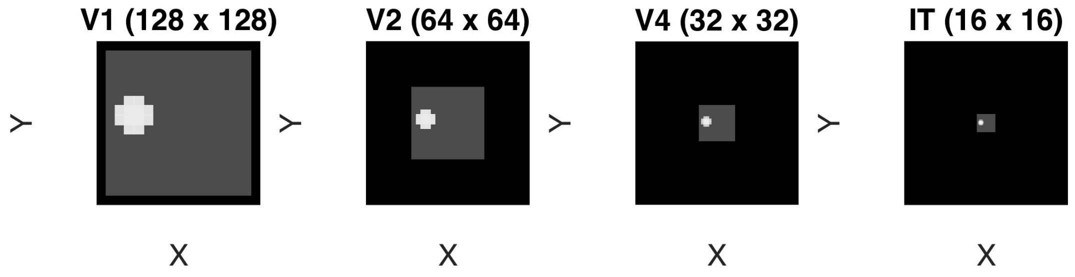


Figure 4.8: A hypothetical AS shown for four layers of the VH. Here, we assume that only location information is captured and the AS is identical for all sheets within a layer.

The method elements involved in priming the VH with an AS are shown in Figure 4.9. Priming at each layer is assumed to occur in discrete time steps with the completion of a priming operation at one layer triggering priming at a subsequent layer. The process begins by first selecting the start layer we wish to prime on *Select Top Layer* followed by loading the AS into tWM (*Load AS to tWM*). Any previously engaged attentional mechanisms are disengaged (*Disengage Attention*), the system checks if an AS for this layer exists (*Is there AS for this layer?*) and then gates the contents to VH (*Gate AS from tWM to VH*). In the case when there is only a partial AS available, priming with the minimum bias strategy is employed in subsequent layers (*Select VH.Prime (minimum bias)*). Here, the minimum bias strategy discussed in Section 4.3.4 is used to propagate the minimum of all feedback bias signals down to the next layer (see Figure 4.13)

4.3.4 VH.Prime without AS

Overview A second method of priming the VH can be achieved by user provided task based information such as "look left" ($VH.Prime(\{'location' :' left'\})$) or "the object is red" ($VH.Prime$ ($\{'color' :' red'\}$)). Alternatively, both pieces of information can be fed into the system at the same time ($VH.Prime(\{'location' :' left'\}, \{'color' :' red'\})$). Here we assume that this information is conveyed to the system by the vTE script executor.

At this time, this method has only been implemented to prime for two kinds of features and four locations. The implemented features it can be primed for are *color* with three values

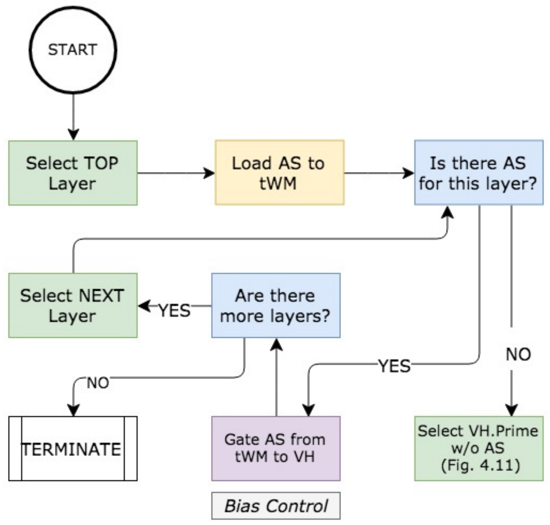


Figure 4.9: VH.Prime with AS Algorithm. Each box is color coded with access to representations (purple), parameter specification (yellow), execute a process (green) and conditional statement decision (blue).

[*red,green,blue*], and *orientation* with nine values: [*-90,-67.5,-45,-22.5,0,22.5,45,67.5,90*]. The locations the system may be primed for are [*left,right,up,bottom*].

Figure 4.10 shows that priming without an AS requires Nodes in the tWM, VH and the use of two NP operations - Bias Control and Bias Feedback. When priming without an AS, we propagate the minimum bias term down each layer of the VH as shown by Equation 4.1. Because of this, only the top most *IT* layer in tWM is used as input.

$$B(t) = \underset{\beta \epsilon \downarrow}{min} \; B_\beta(t) \tag{4.1}$$

Since we model two forms of priming - spatial and feature based, each originating from a different source - two tWM nodes are utilized, indicated as a cascade of tWM AS's in the figure. This separation allows us to apply spatial or feature based priming independently. These two top down signals from the tWM to the VH are combined by taking an average of all converging bias signals as described in Section 3.3.2 for the bias control NP operation. Furthermore, there are also *Intermediate Bias Feedback Nodes* connecting to the VH using Bias Feedback and Bias Control operations. These nodes are also involved in the propagation of the minimum bias term down each layer of the VH.

The involvement of all of these nodes in priming without an AS will be discussed shortly. However, before any of this can occur, the tWM nodes must first be parameterized for a spatial location or feature as discussed below.

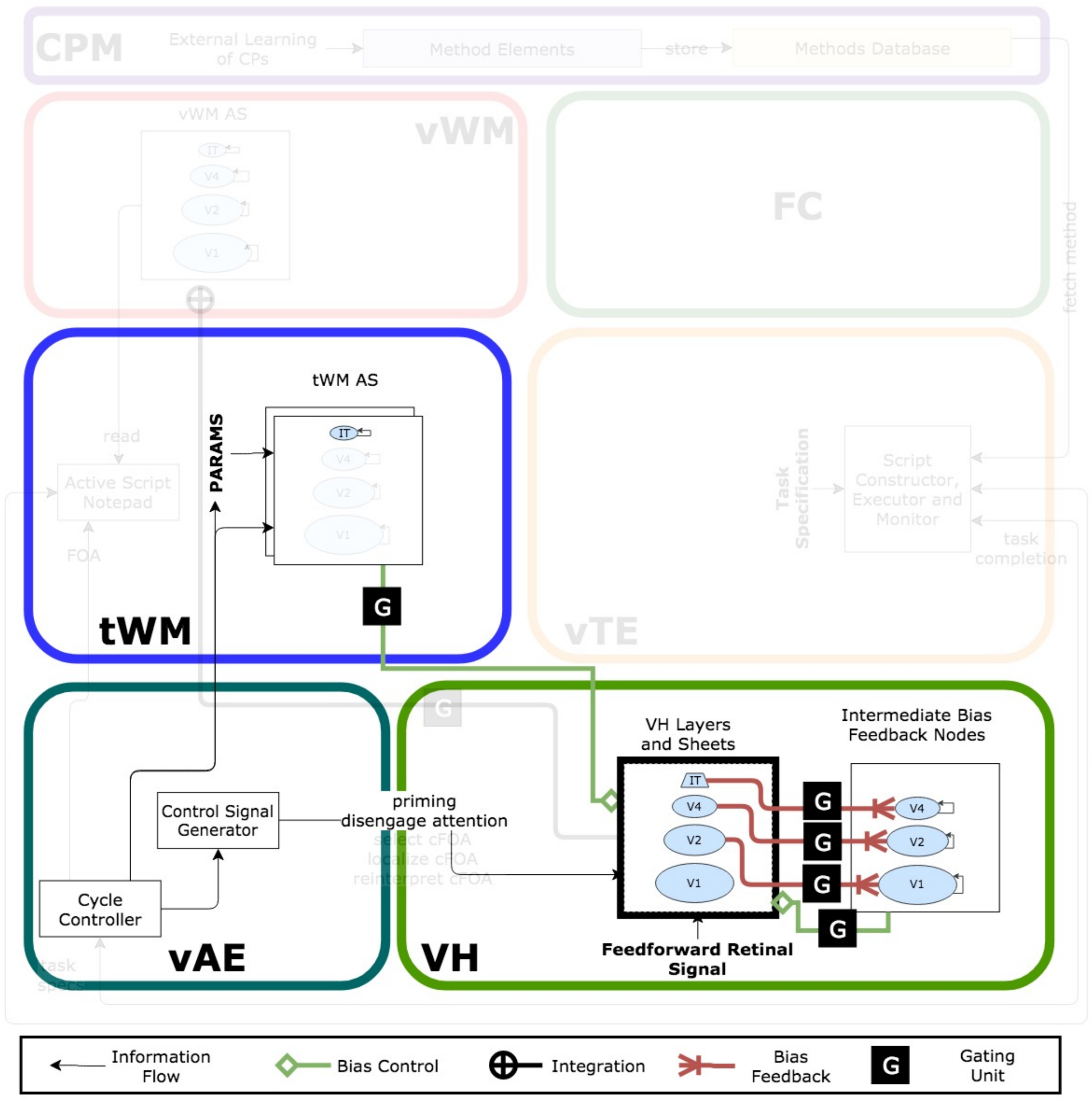


Figure 4.10: Priming without an attentional sample requires task based parameterization that is fed into the tWM. Separate nodes are used for feature and spatial based priming. The contents of the *IT* layer in tWM are gated to the VH using a Bias Control operation. Propagation of the information down the VH requires first computing the minimum bias (see equation 4.1) using the Bias Feedback operation, followed by applying a multiplicative top down bias down the next layer using the Bias Control operation. Note that the Bias Control condensed notation between Intermediate Bias Feedback Nodes and VH Layers and Sheets is included for all layers except *IT*.

Spatial Parameters The location parameterization is uniform across all sheets within a VH layer irrespective of the feature. A specification of the spatial location to prime for is implemented via a Gaussian as shown in equation 4.2. Parameters μ_x and μ_y specify the location the Gaussian is centered while σ is the variance parameter assumed to be uniform across both the x and y dimensions. All parameters were heuristically optimized. Variables $x$and $y$determine how the Gaussian function value changes over the spatial dimensions.

$$P(x,y) = exp(-\frac{(x-\mu_x)^2}{2\sigma^2}) * exp(-\frac{(y-\mu_y)^2}{2\sigma^2}) \tag{4.2}$$

Feature Parameters In contrast, feature priming affects all neurons within a sheet uniformly but the difference in parameterization arises between sheets within the same layer. The input to feature priming is inspired by the work of Martinez-Trujillo & Treue [39]. They note that the direction of motion of the attended stimuli resulted in different neural tuning curves of MT neurons with nearby features showing an enhancement while features farther away showed a suppressive effect. Figure 4.11 shows their findings and indicates that features farther away from the attended one appear to be more strongly suppressed than those that are closer to the attended feature.

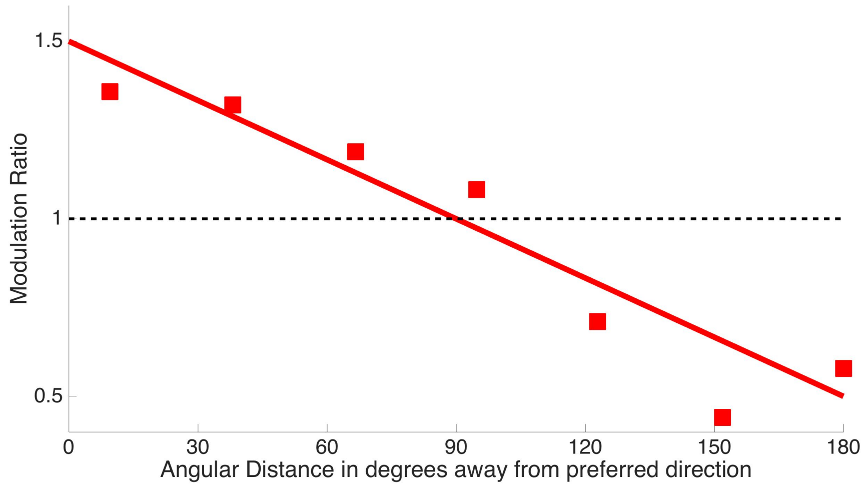


Figure 4.11: Feature based attentional modulation parameterization. The modulation ratio reproduced from [39]

To model these findings for task based parameterization, we introduce equation 4.3.

$$P(d) = \begin{cases} \frac{2\alpha}{L}|d-D_0| + w_{min} & |d-D_0| < \frac{L}{2} \\ \frac{2\alpha}{L}(L-|d-D_0|) + w_{min} & otherwise \end{cases} \tag{4.3}$$

Here d is the feature selectivity of the given sheet and D_0 is the sheet index corresponding to the attended feature. $\alpha = w_{max} - w_{min}$, is a user specified normalization parameter, where $0 < w_{max} < 1$, $0 < w_{min} < 1$, $w_{min} < w_{max}$ and L is the number of dimensions for the feature of interest. For example, assume that we have ten orientations between 0 and 180 with increments of 18° and we wish to attend to the feature with an orientation of 90°. Then, since we have ten feature dimensions, $L = 10$ and the attended feature corresponds to its index, $D_0 = 5$. Figure 4.12 shows how this equation changes with varying values of w_{min} and w_{max}.

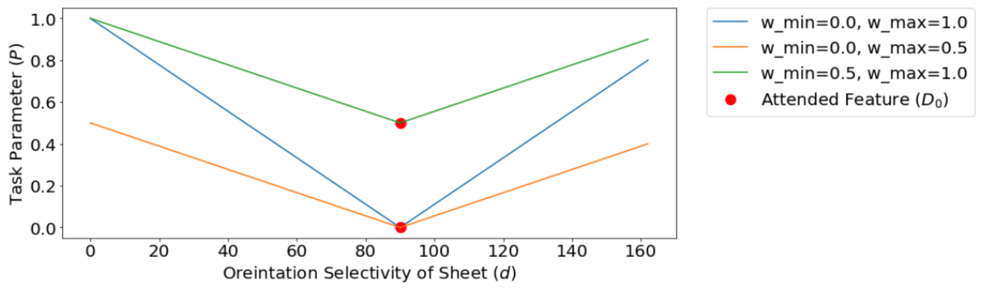


Figure 4.12: Plotting Equation 4.3 for three different pairs of values of w_{min} and w_{max} shows how the corresponding task parameter changes.

Method Elements - Algorithm To apply priming using task based parameters, the method elements shown in Figure 4.13 are required.

The process begins by first checking if a start layer for priming has been specified (*Start Layer Specified?*). This is only used if the method for priming with an AS as discussed in Section 4.3.3 specifies. For example, if there exists a partial AS with information available only for layers IT and $V4$, the $VH.Prime$ without an AS may be called with a start layer specification of $V4$ in order to use the minimum bias strategy to prime lower layers of the VH.

If no start layer is specified, then priming begins at the top layer *Select TOP layer*. Following this, the tWM priming parameters for specifying the feature or location we are searching for are set (*Load (priming params) AS to tWM*) using either either equations 4.2 or 4.3.

Any previously placed attentional mechanisms are disengaged (*Disengage Attention*) and the contents of the IT layer in tWM are gated to the VH (*Gate tWM to VH*) using the Bias Control operation.

To propagate the minimum bias term down to the next layer of the VH, Bias Feedback followed by the Bias Control operation is required.

First, layer L in the VH is gated to layer $L - 1$ of the intermediate bias feedback (IBF) node (*Gate VH to IBF*). When this gating process is enabled, the minimum bias term given by equation 4.1 is also computed and the resulting values are used as the input to the IBF node.

Next, we need a mechanism to now propagate this computed minimum bias to the next layer of the VH and this is accomplished by the Bias Control operation (*Gate IBF to VH*) from layer $L-1$ of IBF to the corresponding $L-1$ layer in the VH.

Each of these gating processes occur at distinct time steps and the process continues down subsequent layers if more layers are present (*Are there more layers?*), otherwise the process terminates.

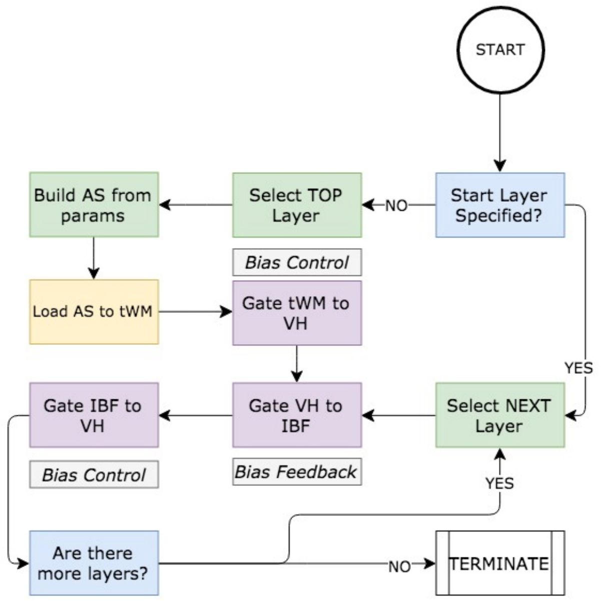


Figure 4.13: VH.Prime without AS Algorithm. Each box is color coded with access to representations (purple), parameter specification (yellow), execute a process (green) and conditional statement decision (blue).

Neural Correlates A variety of cortical areas have been implicated in top down control that influence both behavior and neural response curves given visual tasks. Microstimulation of the dlPFC results in a bias in behavioral target selection [52] while lesions in the PFC have been shown to cause a loss of a top down signal to IT resulting in significant behavioral performance deficits [78]. Inactivation of FEF have been implicated in deficit target detection among distractors in a covert attention paradigm [90] while lesions in LIP negatively impact task performance requiring spatial attention [3]. fMRI studies indicate that several cortical areas have been shown to enhance stimulus and object attributes when attention is deployed to either a spatial '*look left*', feature '*search for the color red*' or an object cue '*search for a car*' [92].

In summary, the neurophysiological literature suggests that several cortical areas are involved in attention and directing attention to a spatial or feature dimension of a visual stimuli results in an enhancement of the corresponding neural responses and improves discriminability of the visual stimuli amongst distractors [89].

The circuits introduced in Figure (4.10) aim to provide this top down control mechanism as a multiplicative bias to layers of the VH. The gating node allows for the timed propagation of the signal down the VH as seen in studies suggesting a latency of attentional modulation down the VH [43]. Spatial and feature based parameterization allow us to account for the difference in neural attention effects due to spatial and feature based attention respectively.

Spatial Priming Example Figure 4.14 shows the bias term activation map computed down layers IT, $V4$, $V2$ and $V1$ after priming is complete. The system has been told to "look left" by the command *VH()− >prime('location':'left');* before stimulus onset and is then transformed into the corresponding parameters using equation 4.2. Lighter areas indicate higher bias values while darker grey areas indicate more suppressed regions.

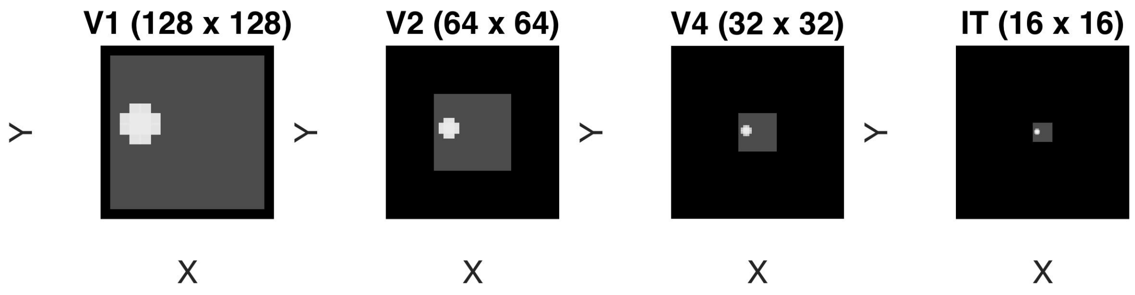


Figure 4.14: Node bias activation map after priming for a spatial location using user specified task parameterization. The activation maps are drawn according to the respective size of the layers.

Feature Priming Example Priming for a specific feature results in a uniform bias across the entire sheet but differences between sheets in a given layer. Figure (4.15) shows three sheets in the same layer with a top down bias imposed on the right most sheet.

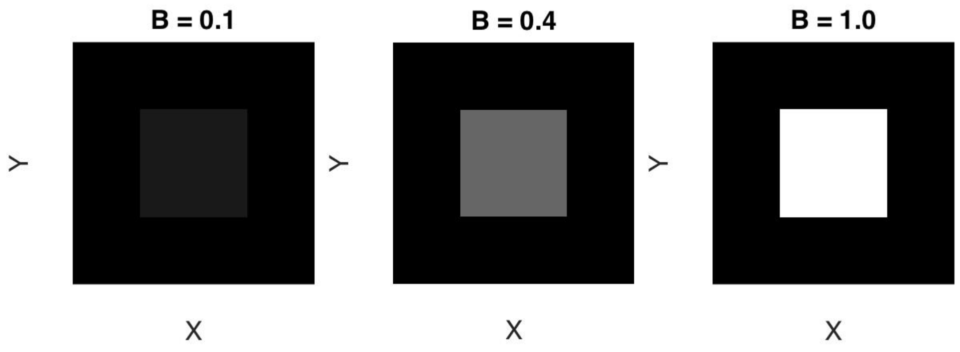


Figure 4.15: Node bias activation map after priming for a feature location using user specified task parameterization.

Combined Priming Example Figure (4.16) shows the effect of combining both feature and spatial priming on the sheet activations in one layer of the VH.

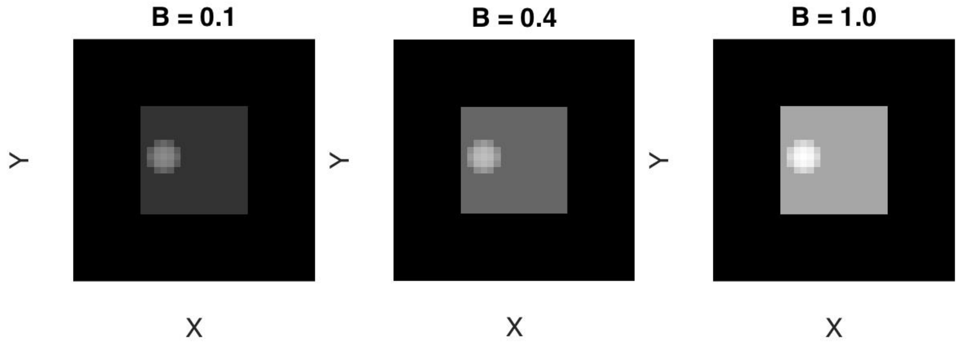


Figure 4.16: Node bias activation map after applying both spatial and feature parameterizations.

4.3.5 VH.Detection

Overview A detection task is one in which one class is noise and the other represents the class of interest. Recall from the literature review that detection is a subset of discrimination. Thus, a single feedforward pass is sufficient for this kind of visual task. This method may be called with an input specification of what we wish to detect (i.e. '*red circle*') or via an attentional sample representing information of the target. Regardless of how this method is called, only information at the top of the VH is available due to insufficient time for a feedback pass. Also note that an AS is only available in this instance if a visual cue is provided prior to the detection task with sufficient time to process its AS. Depending on the specific task demands, there are three kinds of detections we can perform as shown in Table 4.6.

Detection Type	**Description**	**Example Task Instructions**
Location	Detection process only requires comparison of location information irrespective of the features that may appear at that location.	"Press the spacebar if the target appears in the left hand quadrant"
Feature	Spatially invariant detection is performed for the provided feature specification.	"Press the spacebar if the target is red"
Location and Feature	Both the location and feature information are used for detection.	"Press the spacebar if the red target is in the left hand quadrant"

Table 4.6: Three kinds of detection processes may be employed, each of which is governed by the VH.Detection's input specification given task demands.

The detection method requires a complex interaction of the vTE, vWM, tWM and VH as shown in Figure 4.17. Detection allows for the comparison of one class in tWM to the contents in vWM with the aid of *Match Detection* nodes in the vTE. The final decision of whether the match was successful or not is made by a *Decision Node* in the vTE which then sends the result to the Script Monitor to make decisions on the next course of action.

In the VH.Detection, VH.Recognition, VH.Categorization and VH.Identification tasks, the match criterion is defined as the similarity between the AS stored in the vWM to a template AS representing the ground truth in tWM with a detailed description of this process to follow.

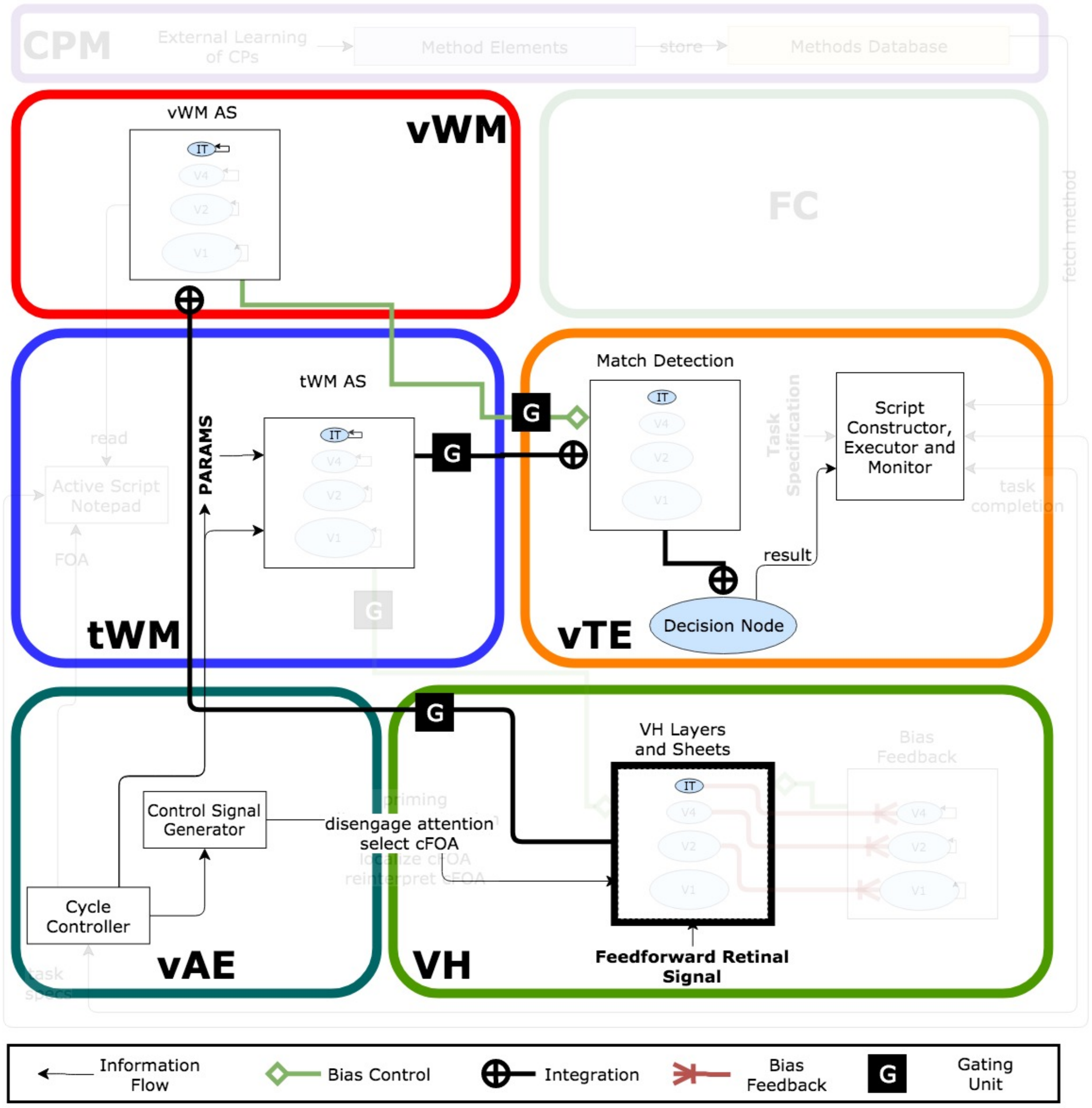


Figure 4.17: Match Detection Network consisting of nodes in the tWM and vWM receiving input from the VH. Contents are compared by the Match Detection Node and a decision node consisting of a single neuron outputs the result. Note that only the top most *IT* Nodes are used for the comparison.

Method Elements - Algorithm To explain how detection occurs in significantly more detail, Figure 4.18 shows the method elements involved.

Before detection can occur, the contents of tWM must first be loaded with a specification of the target (*Load (target) AS to tWM*). If an input specification is provided (i.e. '*red circle*'), then an AS is dynamically built internally. If an AS is provided as input, we assume that it has already been captured by a visual cue presented to the VH prior to the detection task with sufficient time to process its AS using the *VH.Localization* process described in Section 4.3.2. Next, a stimulus input provided to the VH (*Feedforward Retinal Signal*) propagates through the VH and a $\theta - WTA$ selection occurs (*Select cFOA*).

If a winner is not found (*Is the winner found?*), the detection process terminates. Otherwise the *VH.Localization* method is employed to capture the partial AS of the incoming stimulus composed of only the top IT layer (*Select VH.Localization*). Because of the result of this operation, the captured partial AS has now been gated into the vWM with the help of the integration NP operation.

Now that the tWM contains the target template we wish to detect and the vWM contains the input stimulus to which we wish to compare the contents, we must now set some detection criteria (*Set Decision Criteria Params*). This is discussed shortly by introducing equations 4.4 and 4.5. Next, the contents of both tWM and the vWM are gated to the *Match Detection* nodes using integration and bias control operations, respectively (*Gate tWM and vWM to Match Detection*).

This gating of the contents to the *Match Detection* causes the neural activation of the *Decision Node* connected together by an Integration operation to also increase. This *Decision Node* is composed of a single neuron r^{DN} that aggregates all of the neural activations of the *Match Detection* and makes a decision of whether the process was successful or unsuccessful.

Aggregation of the neural activations is done through the Integration operation and roughly resembles equation 4.4. κ is some normalization factor, r_i is the input of the i^{th} neuron in the *Match Detection* node. $S()$ represents the nonlinear transformation of the ST equation (see Equation 3.2 in Section 3.2).

$$r^{DN} = S(\frac{1}{\kappa}\sum_i r_i) \tag{4.4}$$

The decision of whether the detection process was successful (*Decision Criterion Met?*) is computed using Equation 4.5. If the firing rate r^{DN} is within some lower and upper bound dictated by the user specified parameters α_l and α_u respectively, then the operation was successful if and only if it was completed within a certain period of time $[t_l, t_u]$, where t_l and t_u are parameters for the minimum and maximum amount of time taken for the process to complete, respectively. Hence, if the representations differ increasingly more, this will be reflected by a reduction in the firing rate since the cumulative activation of r^{DN} will be significantly less.

$$Decision = \begin{cases} true & \alpha_l \leq r^{DN} \leq \alpha_u \ \ and \ \ t_l \leq t \leq t_u \\ false & otherwise \end{cases} \tag{4.5}$$

The resulting decision is then sent to the vTE for interpretation. The program either terminates if successful, or repeats the process after disengaging attention (*Disengage Attention*) and applying a location based inhibition of return (*Do L-IOR*).

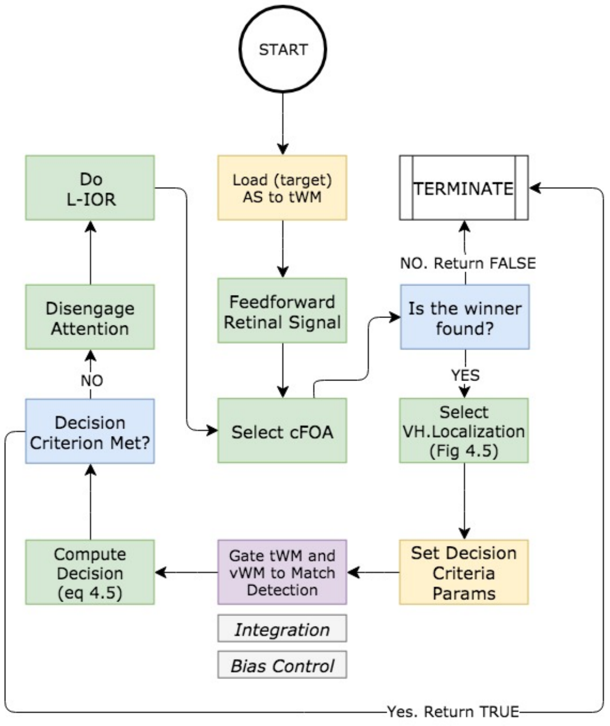


Figure 4.18: VH.Detection Algorithm. Each box is color coded with access to representations (purple), parameter specification (yellow), execute a process (green) and conditional statement decision (blue).

Neural Correlates The neuroscience of decision making is vast and the underlying brain networks and circuitry even more so. In the context of decision making for stimulus selection, a combination of task rules, prior expectations and the current state of an organism all influence the choices and actions an organism may make. These influences, implemented by a variety of cortical and subcortical networks, feed information to other cortical areas responsible for integrating and implementing attentional selection [92].

The generic process by which decision making occurs on visual tasks may be divided into three stages: gathering sensory evidence, integrating information over time and criterion selection [8]. The gathering of the sensory evidence occurring in the visual cortex is responsible for encoding information about the object, feature or spatial location of interest. The lateral intraparietal cortex (LIP) and the FEF are some areas proposed for this accumulation of "sensory evidence" over time to improve the signal to noise ratio [8]. Decision criterion selection is thought to occur in both cortical and subcortical areas with a specific emphasis on the basal ganglia (BG) reflecting action selection rules [4] and implementing a winner take all (WTA) algorithm. The basic mechanism by which the BG exerts control over other areas is through a process of disinhibition [54].

For example, the orbital frontal cortex (OFC) is active in the anticipation of rewards [68] while the frontal eye field (FEF) and the intraparietal cortex are implicated in implementing selection mechanisms [92]. Other portions of the prefrontal cortex including the lateral prefrontal cortex (lPFC) have been shown to reflect an animals decision of the selected target [23]. Neurons in the PFC may also play a role of biasing activity in favor of behaviorally relevant stimuli in part by providing feedback bias signals to the inferior temporal (IT) cortex [45].

The temporal component of integrating information over time is also evident from neurophysiological studies. As an example, macaque monkey FEF neurons respond indiscriminately to the target or distractor on a visual search task at first but later the neural activity evolves to signal the location of the target stimulus [68]. A higher visual similarity results in neurons that show higher firing rates for distractors that closely resemble the target features resulting in more behavioral errors on the visual task [68]. Further evidence shows that eye movements occur when some movement related neural activity in FEF increases above some threshold while a fixation neurons activity decreases by a sufficient amount indicating that competitive mechanisms for action selection may be at play.

In our detection method, there is no cortical area that is explicitly defined with respect to the discussed STAR modules. Instead, this neurophysiological evidence is used only to provide important clues for what the computational counterparts of decision making may be, irrespective of the corresponding cortical areas involved.

The nodes and the corresponding NP operations connecting them as shown in Figure 4.17 were inspired by these computational counterparts from neuroscience by implementing gathering of sensory evidence, integrating information over time and criterion selection. Prior expectations of the stimulus are encoded by "templates" in the tWM representing the target we wish to find while task rules are encoded by the vTE itself. The gathering of sensory information occurs through the VH which computes the features and relevant locations given some input stimulus. This information is moved to the vWM via the Integration NP in preparation for comparison to the contents of the tWM. Here, the tWM and vWM connect to the *Match Detection* node via an *Integration* and *Bias Control* operation implementing a multiplicative comparison of the two signals. The *Match Detection* and *Decision Node* implement integrating evidence over time while the decision criterion for criterion selection is defined by equation 4.5.

Example Figure 4.19 shows the response of the *Decision Node* (r^{DN}). The plot is of the same single neuron under two different conditions: when the target is validly or invalidly cued during the Egly & Driver 1994 experiment [19]. At $t = 0$, the contents of vWM and tWM are gated to the *Match Detection* node. The lower and upper bounds $t_l = 2$ and $t_u = 4$, respectively, are indicated by dashed black vertical lines. Notice that when $t > t_l$, the system begins to distinguish a *Valid Cue* from an *Invalid Cue*. At $t \geq t_u$ there is a significant different in the neural activation in the two conditions. Hence, these values were heuristically chosen for our detection process. Since the valid cue condition has a higher neural activation, it reaches a threshold for detection earlier. This is reflected in a faster reaction time output of the system in the valid cue condition, consistent with the findings in the original experiment. Further discussion of this experiment is provided in a subsequent section. Note that each time step of a simulation equals approximately 2.5 ms in real neural time.

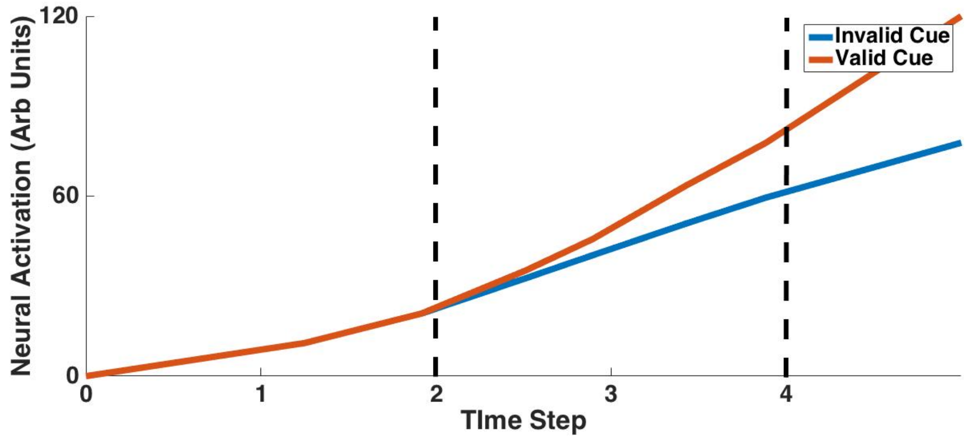


Figure 4.19: Plot of the neural activations of the r^{DN} match decision node upon presentation of a valid or invalid cue. Vertical lines indicate the t_l and t_u time bounds for detection. One time step is approximately equal to 2.5 ms in real time.

4.3.6 VH.Recognition, VH.Categorization and VH.Identification

Overview This section extends the Match Detection operation presented in the previous *VH.Detection* section to allow it to be usable for multiple classes. As with the Detection operation, the Recognition, Categorization and Identification methods rely only on a single feed-forward pass requiring about $150ms$ to complete [85]. Each of these methods rely on the same underlying implementation, but differ in the kind of response they output.

A recognition task is one in which neither of the two classes are noise. An example task may provide instructions "Press 'x' if you see a circle, press 'z' if you see a square". A categorization task is one in which the system connects each stimulus to some category. For example, given three easily separable categories - chairs, buildings and cell phones, the task is to respond indicating to which class the stimulus belongs. In the identification task, the system assigns a different response to each stimulus. For example, if we have an array of letters from *A-Z* as visual stimuli, each one needs to be mapped to a different class.

These three methods require that we introduce additional nodes in the tWM and vTE to accommodate the extra classes we wish to recognize, categorize or identify. In Figure 4.20, there are three nodes each for tWM, *Match Detection* and *Decision Nodes.* The method elements used for this process are identical as those in the *VH.Detection* method except that here, comparison against multiple templates is occurring in parallel. As such, the *Load (target) AS to tWM* command in Figure 4.18 is applied for each class template we are comparing against, as discussed below.

Method Elements - Algorithm One major difference between *VH.Detection* and the methods described here is that now we have the existence of more than one class. Thus, equation 4.5 must be modified such that it returns the neural activation of the *Decision Node* rather than a boolean value as shown in equation 4.6. The differences in the two equations are bolded.

$$\mathbf{r_i^{DN}} = \begin{cases} \mathbf{r_i^{DN}} & \alpha_l \leq r_i^{DN} \leq \alpha_u \;\; and \;\; t_l \leq t \leq t_u \\ \mathbf{0} & otherwise \end{cases} \tag{4.6}$$

Now, if we have multiple templates in the tWM comparing to the contents of vWM in parallel, then we will get an array of *Decision* values. Applying the softmax equation 4.7 on this output results in a probability distribution over the N possible classes.

$$P(j) = \frac{r_j^{DN}}{\sum_{i=1}^{N} r_i^{DN}} \tag{4.7}$$

In addition, for one of the experiments that will be discussed later (see Section 6.1.3 and Appendix (B)), we introduce a *Memory Consolidation* node. The role of this node is to prevent new stimulus from interfering with identification processes that may be initiated early on by inhibiting the VH *IT* layer using a Bias Control operation.

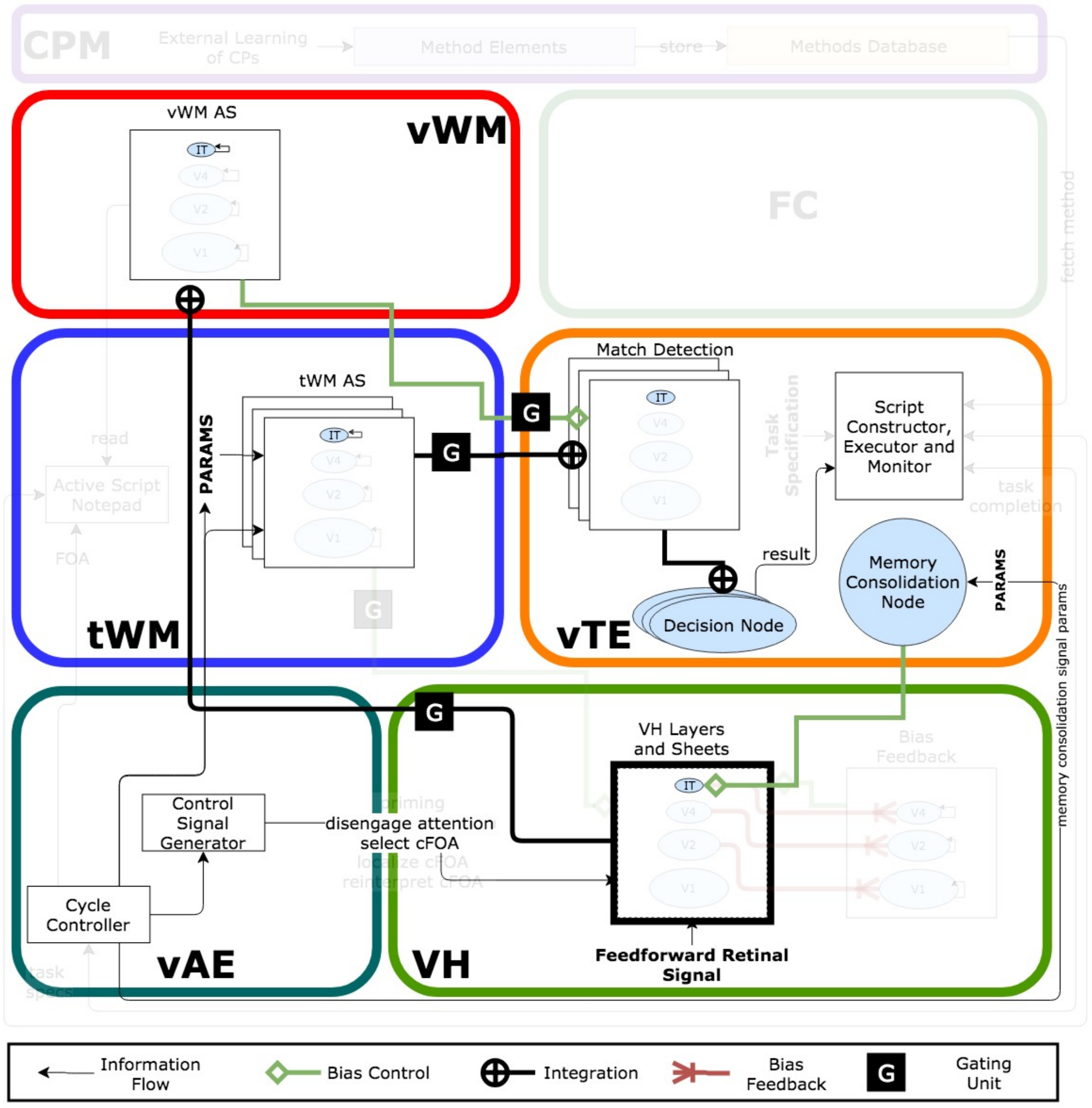


Figure 4.20: Match Detection Network consisting of nodes in the tWM and vWM receiving input from the VH. Contents are compared by the Match Node and a decision node consisting of a single neuron outputs the result. Note that only the top most *IT* Nodes are used for the comparison.

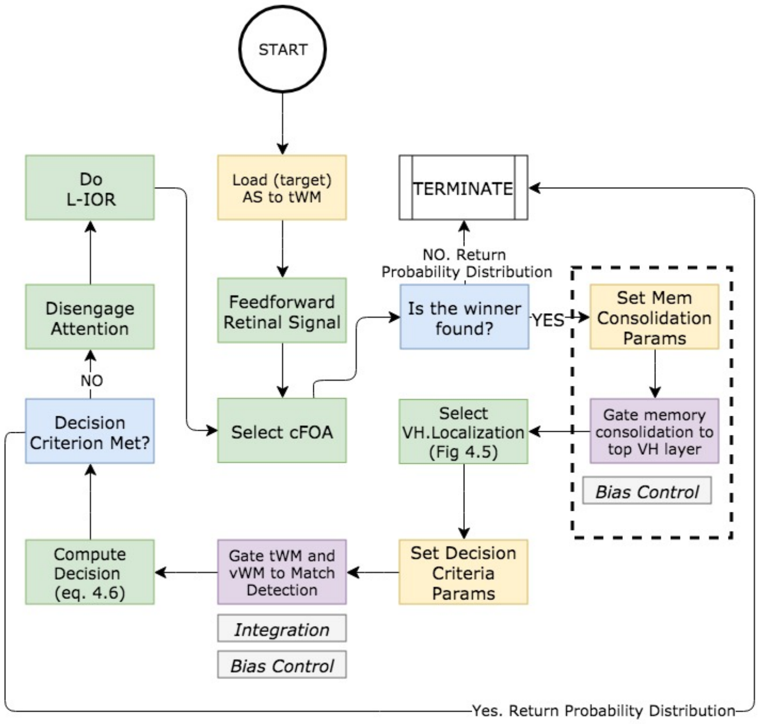

START
Load (target) AS to tWM
Feedforward Retinal Signal
Select cFOA
Is the winner found?
NO. Return Probability Distribution
TERMINATE
YES
Set Mem Consolidation Params
Gate memory consolidation to top VH layer
Bias Control
Select VH.Localization (Fig 4.5)
Set Decision Criteria Params
Gate tWM and vWM to Match Detection
Integration
Bias Control
Compute Decision (eq. 4.6)
Decision Criterion Met?
NO
Disengage Attention
Do L-IOR
Yes. Return Probability Distribution

Figure 4.21: Method elements for classification when there is more than one class, neither of which are noise. Each box is color coded with access to representations (purple) and parameter specification (yellow). The addition of the memory consolidation parameterization and gating indicated by a dotted box around the corresponding components distinguishes this method from that of Figure 4.18.

Example Figure 4.22 shows a plot of two different match decision nodes during a recognition process.

Let's assume that r_1^{DN} and r_2^{DN} are decision nodes representing the evidence accumulation for two classes - a circle and square, respectively.

Looking at the figure, the approximate neural activations are $r_1^{DN} = 60$ and $r_2^{DN} = 70$ at $t = 4$. Then, our softmax function returns $P = [0, 1.0]$ according to equation 4.7. Taking the largest value, the neuron r_2^{DN} indicating that a 'square' is most likely recognized is given as output to the Script Monitor in the vTE. Although we have illustrated how recognition works, this example can be extended to any number of decision nodes N such that we can apply this process to any number of classes if required by VH.Categorization or VH.Identification methods. This simple example indicates that the process of computing equations 4.6 and 4.7 can be used to indicate which class is most likely recognized, categorized or identified.

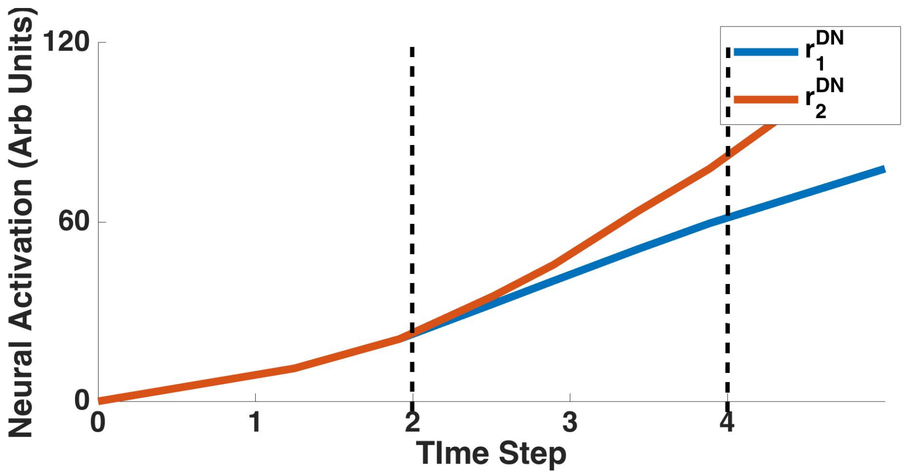


Figure 4.22: Plot of the neural activations of the r_1^{DN} and r_2^{DN} match decision nodes when the target symbol '=' is presented to the system. One node shows higher selectivity than the other. Vertical lines indicate the t_l and t_u time bounds for detection. One time step is approximately equal to 2.5 ms in real time.

4.3.7 VH.Identification (Within Category)

Overview The within category identification task invokes a recurrent binding strategy when more detail about a stimulus is required [85]. In other words, when information at the top most layer is insufficient to perform a match operation, the system must traverse down to lower layers of the visual hierarchy for more information. Within category identification takes an additional 65 ms and is prone to errors since the process may be interrupted with changes in incoming visual stimuli. Hence, identification may require partial or full recurrence binding.

Note that the within category identification process discussed here is different than the identification process described in Section 4.3.6 requiring only a single feedforward pass. The within category identification process is included for completeness but the implementation is left for future work.

The major difference here is that information available in lower layers of the VH is also used to come to a decision on the targets identity. Hence, the *Select VH.Localization* command in Figure 4.21 is used to retrieve an AS containing more than just the *IT* layer of the VH. Equations 4.6 and 4.7 are still used for determining whether the identification process is successful.

4.3.8 FC.setFHMBias

Overview The fixation history map (FHM) as part of the fixation controller (FC) is intended to bias against revisiting previously seen locations but can be overridden by task demands [29]. It does this by storing the last several fixations each decaying over time, hence implementing a built in location based inhibition of return (L-IOR). The representation is gaze centered, larger than the visual field and contains a sequence of recent fixations. As an example, if we wish to maintain attention within a 2 degree area within the visual field, a task bias can suppress regions outside of this such that no eye movements occur.

Behaviorally, a L-IOR enhances or impairs speed and accuracy of object detection depending on when a target appears with respect to a cue and whether the target was validly or invalidly cued [56]. If the targets appeared 100-300 ms after stimulus onset, then speed and accuracy of object detection is enhanced while targets appearing at a later time; 300-3000 ms showed the opposite effect

This method sets the appropriate FHM top down (TD) bias used to explicitly influence the FCs next eye movement given task demands. Hence, the role is to provide a task specific attentional pull [86] triggered via an exogenous signal with influences from higher cortical areas. An in depth review of the fixation control strategy within STAR can be found in [86].

Figure 4.23 shows that several representations residing in the FC and vWM are required for biasing the FHM and we summarize this below.

1. *FHM Bias Unit*: This node is used to determine the degree of influence of a top down bias. We can use this to completely ignore previously seen locations and rely only on task demands. This may be the case if the system is told "hold your fixation at the centre of the screen regardless of what you see".

2. *Bias Selection TD/BU*: The TD and BU representations have excitatory and inhibitory weights, respectively. These representations are manipulated by the *FHM Bias Unit* and are also involved in determining the degree of influence of a top down bias.

3. *FHM TD/BU*: The endogenous FHM (FHM_{TD}) is a context input node reflecting task specific attentional pull mechanisms. It takes in user specified parameters such as "look at the center fixation" and this may then be converted to a corresponding AS (refer to equation 4.2 in Section 4.3.4). The endogenous FHM (FHM_{BU}) is assumed to be generated from the FC strategy of STAR (see [86]).

4. *Combined FHM*: This representation combines the FHM_{TD} and FHM_{BU} using an integration operation after a top down task bias has manipulated their corresponding representations.

Method Elements - Algorithm The process for biasing the FHM in order to influence L-IOR occurs in several steps.

First a task bias for a given location must be specified (*Set FHM task bias for location*) using an AS as the input to the FHM_{TD} nodes as described earlier.

Next, a task bias weight indicating the degree of influence for top down and bottom up information is specified (*Set FHM task bias for weight*) using the *FHM Bias Unit* to the *Bias Selection TD/BU* nodes. The intuitive way to understand this is that as the input neural activation from *FHM Bias Unit* increases, the *Bias Selection TD/BU* will increase/decrease as well. Namely, if the neural activation of the bias unit is high, top down influences of where to look will be greater than bottom up influences. This bias then effects the FHM_{TD} and FHM_{BU} representations since they are connected to the *Bias Selection TD/BU* nodes via a Bias Control operation.

FHM_{TD} and FHM_{BU} are then gated to the combined FHM (*Gate TD/BU FHM to combined FHM*) and this information is used to influence the next FOA that may be selected by the WTA algorithm. In our TarzaNN system implementation, the *Combined FHM* Node is simply implemented as a linear decay over time and does not incorporate the other elements of the FC.

In the current implementation, none of the other components described in [86] are integrated as that is beyond the scope of this thesis. Instead, the *Combined FHM Node* is implemented as a simple weight matrix that indicates that a winner is unlikely to be selected at locations where a L-IOR signal is active.

Neural Correlates The neural correlate of the FHM are thought to be in the FEF since it has been shown to be an important area for visual working memory representation of both retinal and extra-retinal space [86]. The FHM is also responsible for combining the recent fixations with task specific biases [86]- where then may such a task bias originate from? A possible candidate

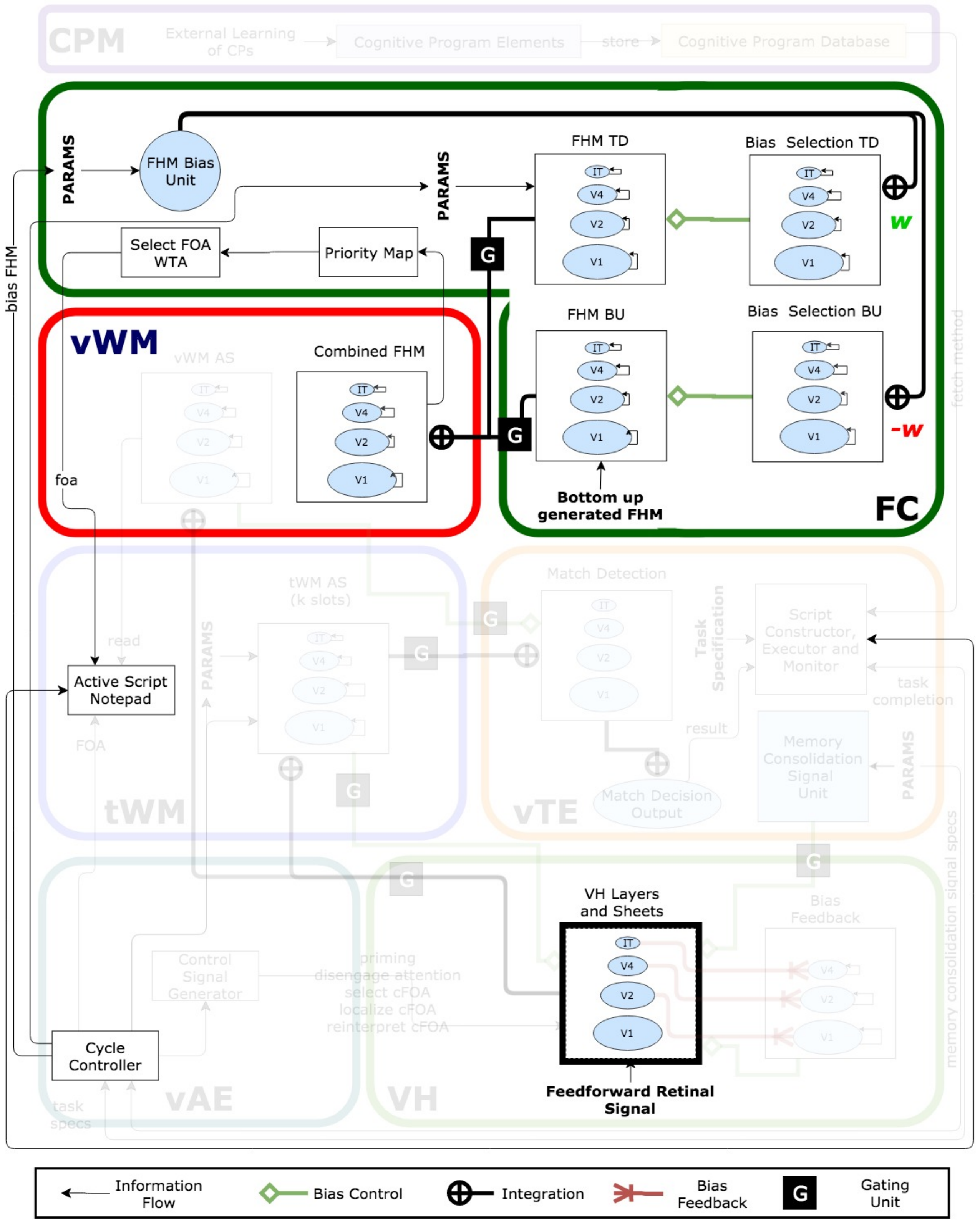


Figure 4.23: Network unit operation required for biasing the FHM.

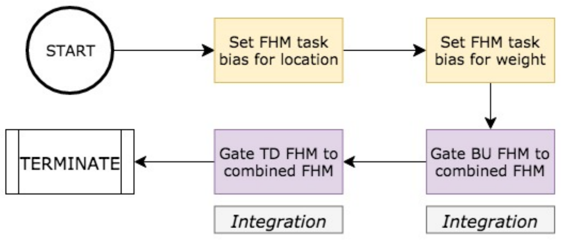

START
Set FHM task bias for location
Set FHM task bias for weight
Gate BU FHM to combined FHM
Gate TD FHM to combined FHM
TERMINATE
Integration
Integration

Figure 4.24: Fixation History Map Algorithm. Each box is color coded with access to representations (purple), parameter specification (yellow), execute a process (green) and conditional statement decision (blue).

is the LIP since stimulation here can lead to covert and overt shifts of attention [3]. However, the dorsolateral prefrontal cortex (dlPFC) is a more promising candidate as stimulation here has been shown to bias saccade target selection potentially by modulating the balance between excitation and inhibition [52]. The lPFC among many other PFC areas are also implicated in flexible shifts in attention and to encode attentional shift rules [92].

Thus we propose that a top down control signal may be responsible for modulating the FC by providing a task bias that shifts the excitatory and inhibitory balance of the FHM. This shift may then be reflected as our attentional pull mechanism as shown in Figure 4.23.

Example To demonstrate the competitive endogenous and exogenous mechanisms, Figure 4.25 plots the activation of one neuron in the FHM_{TD} and FHM_{BU} as a function of the input task bias (shown as *bias FHM* in Figure 4.23) provided to the *FHM Bias Unit*. Note that the values have been normalized for clarity and all other input values and parameters are kept constant.

If the neural activation of the *FHM Bias Unit* is gradually increased, it will cause a corresponding increase and decrease of the TD and BU *Bias Selection* nodes, respectively. Since these bias selection nodes are connected to the FHM_{TD} and FHM_{BU} using a Bias Control operation, this will result in a multiplicative scaling of the representations.

Here, we notice that we can vary the task influence by modulating the task bias. As the value of the task bias increases from 0.0 to 1.0, task dependent endogenous influences dominate bottom up exogenous sources of attention.

4.4 Summary

We began this section with the aim of proposing a framework that bridges the gap between Neural Primitives (NP), Cognitive Programs (CP) and the rest of the STAR architecture. In doing so, the Cognitive Programs Memory (CPM) was divided into Method Elements (CPM-ME)

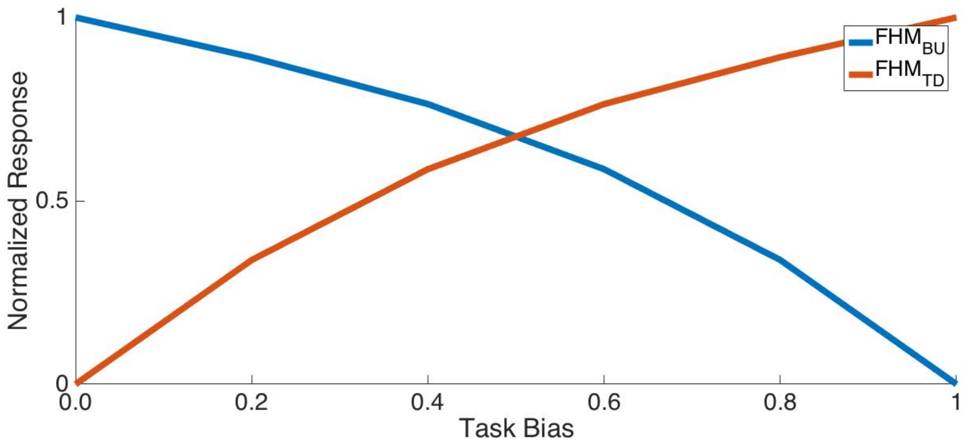


Figure 4.25: Normalized response of FHM_{BU} and FHM_{TD} maps when the task bias is varied between [0.0,1.0].

and the Cognitive Program Database (CPM-MD). The CPM-ME contains elements required to assemble CPs as methods. Methods are then stored into a database-like structure called the CPM-CPD, where they can be retrieved, parameterized and executed by the vTE.

In this thesis, eight implemented CP methods in the CPM-CPD were introduced - VH.Localization, VH.Prime (with and without AS), VH.Detection, VH.Recognition, VH.Identification (within and between category) and FC.setFHMBias. Each of these methods were detailed in terms of the representations they act on, the NP computations required and the numerous Method Elements required for execution. Table 4.7 summarizes the Method Elements used to compose the methods in our Methods Database.

Method Elements		**Description**
Queries	Conditional Statements	Reads the current state of the system with the result used to decide on the next action. Uses conditional IF-THEN-ELSE logic.
Actions	Access to Representations	Reads the current neural activation of a representation. Enables or disables NP Gating operation.
	Parameter Specifications	Includes specification of control signals, parameterization of NPs, context Nodes and setting decision criteria.
	Executes Processes	Executes internal processes (i.e. Disengage Attention, Engage L-IOR, θ-WTA selection, etc.)

Table 4.7: The Method Elements used to compose Cognitive Program Algorithms.

All in all, this framework provides prebuilt functions for executing a variety of visual tasks without the need to construct new ones on a task by task basis, opening the way for rapid development of the STAR architecture.

5 Framework Implementation

5.1 Overview

In this section we detail the implementation of the Neural Primitive and Cognitive Programs Memory framework within the pre-existing TarzaNN system that was previously developed in the lab [64]. Since then, several performance improvements and a complete restructuring of the code has lead to subsequent versions. TarzaNN3.0 is the currently maintained version. Since the implementation of the work in this thesis is so heavily integrated into TarzaNN 3.0, this section first introduces the major constituents of TarzaNN 3.0

TarzaNN is a simulator for visual attention modeling that implements a subset of the functionality of STAR. It was designed in C++ - an object oriented programming language that consists of classes which act as blueprints for defining objects. Classes may inherit from one another, for example, a class *Car* may inherit from *Vehicle*. The system was implemented in C++ since it is a fast, powerful and efficient language. TarzaNN3.0 contains a complete implementation of the visual hierarchy including feedforward stimulus processing and feedback processing incorporating the WTA algorithm. Furthermore, definitions of key control signals such as selection of a winner, disengaging attention, applying suppressive surround, applying inhibition of return or setting $\theta - WTA$ selection parameters are included and interact directly with the *AttentionExecutive* class to implement a *visual attention executive* (vAE).

This existing framework provides an excellent starting point since some of the key computations required by Cognitive Programs are already present. Furthermore, since the STAR model requires several components including the vTE, vAE, tWM, vWM, VH, FC as well as algorithms for fixation and selection, developing such a system from scratch would be beyond the scope of this thesis.

Figure 5.1 shows several of the classes implemented. Classes are color coded in white and blue representing abstract and concrete classes, respectively. The *CPC* class representing the Cognitive Program Compiler used for generating the task specification is also concrete but color coded purple to indicate that it is external to this thesis. See [32] for more details on its implementation. An abstract class contains at least one abstract method that needs to be implemented while a concrete class has all functions implemented. Notice that in the figure, the *NODE* and *NeuralPrimitive* classes are abstract since their implementation is dependent on the kind of Node - Default, Context or Memory (see Section 3) and the kind of Neural Primitive computation - Bias Control, Bias Feedback and Integration. Each Neural Primitive is represented in its own concrete class (not shown in the figure), the details of which will be described in subsequent sections.

Connections with diamonds indicate “has-a” relationships. For example, the abstract *NODE* and *NeuralPrimitive* classes have several classes that inherit from them as discussed in more detail in the next section. On the other hand, a “has-a” relationship can be seen by the *TN3Network* which contains Nodes, Neural Primitives and other TN3Networks.

The next two sections detail the major components of Figure 5.1 as summarized below.

1. *Neural Primitives*: This section outlines the implementation details of NODES and Neural Primitives as formalized in Section 3. Importantly, it outlines how the Neural Primitive operations are applied onto NODES to implement the corresponding computations given by the equations summarized in Table 3.3. Neural Primitives are implemented via the *tn3network* package shown in Figure 5.1.

2. *Cognitive Programs Memory*: This section explains how Method Elements (CPM-ME) - the NP operations from the *tn3network* package and the Base Methods from the *tn3cpm* package can be combined to form stored methods. How these methods are retrieved, parameterized and executed by the **TN3Runner** and several components it interacts with is also discussed.

5.2 Neural Primitive

5.2.1 Overview

In Section 3, the Neural Primitive Specification details a set of computational elements that are inspired by neural computations and their correlates. One of the main goals of this chapter was to outline a mathematical formulation of these computations as summarized in Table 3.3.

The implementation of the Neural Primitive (NP) framework is composed of representations as *NODES* and operations as *Neural Primitives* that are encapsulated into a *TN3Network*. Figure 5.2 shows these components and their implemented classes with a subset of the implemented methods and variables listed. For these methods and variables, the return type is indicated - with most function calls returning *void* indicating that no value is returned. Some methods, such as the *Neural Primitive* have a return type of *virtual void* indicating that the function needs to be implemented by a concrete class inheriting from it.

Each of the *TN3Network*’s reside in one or more of the STAR modules (VH, vTE, vWM, tWM and FC) and are simply a way to group related Nodes and NPs together. This provides ease of accessibility and functional segregation from other networks. For example, the priming

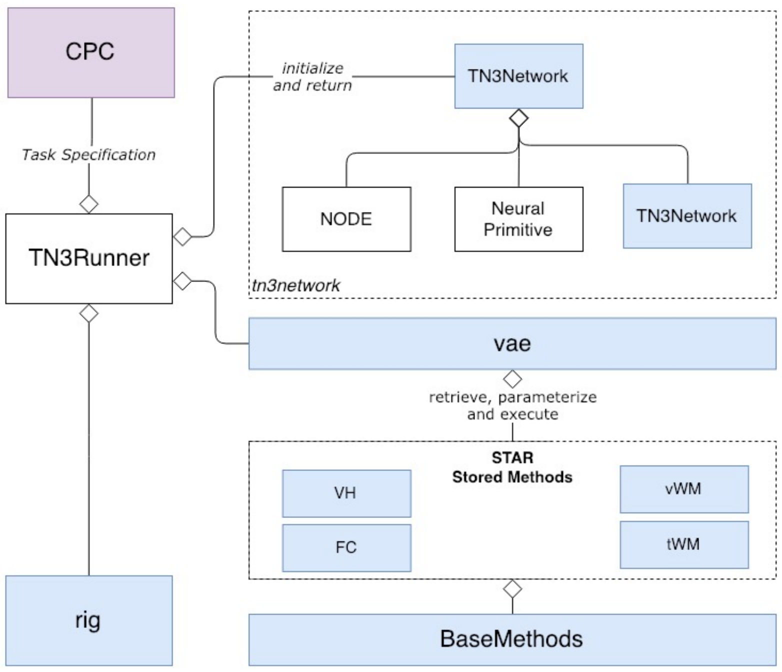


Figure 5.1: Implementation of the CPM framework requires several components with the *TN3Runner* being central to task execution. Classes in the *tn3network* implement the Neural Primitive Specification (see Section 3). The remaining classes, excluding the CPC implement the Cognitive Programs Memory Specification (see Section 4). Diamonds indicate "has-a" relationships. Diamonds between groups of items indicate that each item in the group has a "has-a" relationship with the corresponding class it is connecting to. Classes in white and blue boxes are abstract and concrete, respectively.

network with an attentional sample (AS) described in Section 4.3.3 has a network name "PrimeWithAS" and contains the corresponding Nodes and NPs shown in Figure 4.7. All of these classes and their implementation exist within the *tn3network* package.

A *TN3Network* is implemented as a directed graph with each *NODE* representing a set of vertices V and edges E being represented by the *Neural Primitive* operations. There exists a global network $G = (V, E)$ and subnetworks $G_i = (S, T)$ where $G_i \subset G$, $S_i \subset V$ and $T_i \subset E$. Each network contains Nodes, NPs and other networks. Since each NP operation acts to transform a destination *NODE* given some source *NODE*, a directed graph is useful for keeping track of this directionality.

Section 5.2.2 describes how a *TN3Network* can be created, initialized with all of its elements added to form a graph that can be stored for easy retrieval later. After graph creation, Section 5.2.3 describes how the nodes are updated by computing the corresponding NP equations described in Table 3.3.

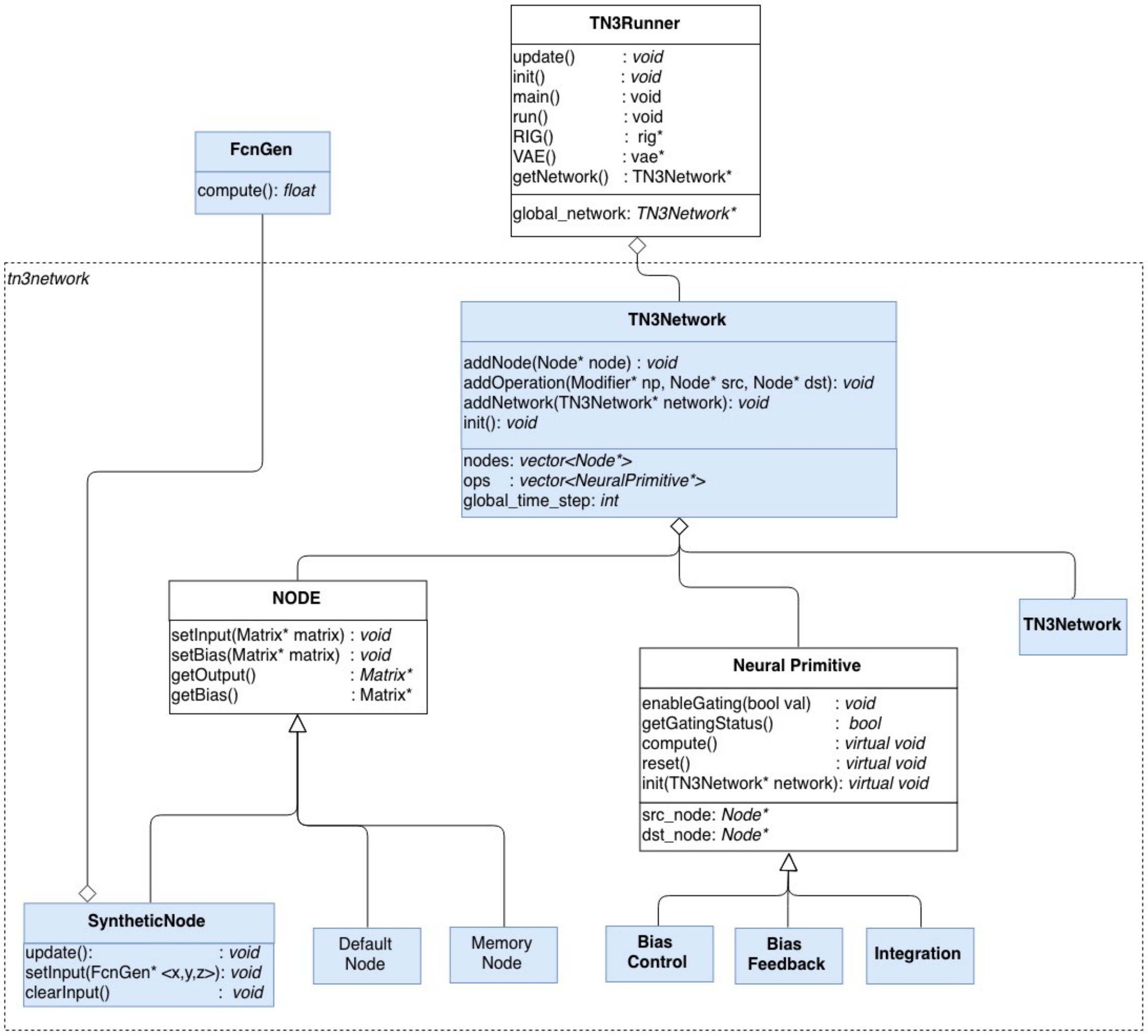


Figure 5.2: A TN3Network is a graph structure with Nodes as vertices and Neural Primitives as edges. Subgraphs may also be present as indicated by the TN3Network on the far right. Arrows and diamonds indicate inheritance, also known as “is-a” and “has-a” relationships, respectively. Classes in white and blue boxes are abstract and concrete respectively.

5.2.2 Graph Creation

Before we can perform any NP computations, we first need to create a Graph consisting of **NODES** (vertices), **NP Operations** (edges) and subgraphs (or subnetworks). Subgraphs are a way of providing functional segregation from other networks to make the code easier to understand as discussed earlier. This stage is used to create all the Nodes in the various STAR modules such as the vWM and tWM as well as to define how NPs will be used to connect them together. Once all of the elements are added, the network is then instantiated.

When executing any visual task using the provided CPs, there needs to be a specification of the entire network or graph used for accessibility of Nodes and NPs. The *TN3Runner*, discussed in more detail in later sections and shown in Figure 5.2 contains a reference to this network - referred to as the **global network** implemented as a *TN3Network***.** For example, the complete STAR architecture as shown in Figure C.1 in Appendix C has Nodes and NPs that all reside in this **global network**.

The *TN3Network* class contains the necessary functions for adding Nodes and NP Operations as well as a function call to setup the network once it has been fully specified. Among the many functions in this class, there are five important ones for graph creation.

1. *TN3Network(string label)*: A constructor for creating a new network with a specified name.
2. *addNode(Node* node)*: Adds the Node to the network.
3. *addOperation(NeuralPrimitive* np, Node* src, Node* dst)*: Adds a NP operation to connect two existing Nodes in the network.
4. *addNetwork(TN3Network* network)*: Adds subnetworks to the network.
5. *init()*: Initializes the network and prevents further modification of the network. Also performs error checking to ensure network validity. For example, network names must be distinct.

In the next few sections we explain how *Nodes* and *Neural Primitives* are created before they can be added to the network using the *addNode()* and *addOperation()* operations followed by how the network is initialized before it is ready for computing the NP operations using the *init()* function.

Vertex (NODE) Creation

A representation is implemented as a matrix of neurons encapsulated in a **NODE** class. It is parameterized with a specification of the *node name*, the size of the matrix of neurons

$(width, height)$ and the *neuron type* provided in its constructor. The *neuron type* is set to *ST_NEURON* unless otherwise specified, but may also be set to *LINEAR_NEURON*. This modularization allows for testing of different neuron types in the experiments and also separates the neural encapsulation from its underlying neural equation computation.

Each Node is represented by the equation presented in Section 3 and rewritten in equation 5.1 below for reference, where R_A, B_A and F_A are the neural activation, bias and feedforward matrices of a Node with the name "A", respectively. The neural activations and the bias term matrices at the current time can be returned by the function calls $getOutput()$ and $getBias()$, respectively. The three kinds of Nodes shown in Figure 5.2 are the implementations of the abstract **NODE** class.

$$R_A(B_A, F_A) \tag{5.1}$$

A **DEFAULT NODE,** such as those used in the VH and the match detection module as shown in Figure C.1 in Appendix C, computes the neural activations of all neurons at each time step. Additionally, to allow for compatibility with the previous implementation of the *Sheet** class in the *TarzaNN3.0* framework, we can wrap a sheet into a default node. The *Sheet** class was originally used for representing neural activations within the *VH*. However, this class does not provide all the functionality we require for NODES. To extend their functionality, we can use a default node as a wrapper around the sheet with the constructor *DEFAULT_NODE(Sheet* vh_sheet)*. Notice here that no specification of the node name or the size of the matrix is necessary. These parameters are automatically extracted from the *Sheet**. This wrapper also allows for a uniform representation of matrices of neurons irrespective of the module they reside in. As such, it will be useful when performing operations on these Nodes, such as priming the VH.

A **MEMORY NODE** maintains its neural activation when an input is provided for the first time. The vWM, tWM and FC all contain memory nodes. The implementation of this node still computes the neural activation governed by the ST equation. However, the feedforward input to the node, F_A is kept constant even if the inputs to this node change over time. Thus, this node will maintain its activation even if the feedforward input is removed at a later point in time. To ensure the memory node is versatile such that we can update its neural activation to store different information when required, the $resetMemNode()$ function is used. After this function is called, the neural activations will be updated and maintained when any of the NP operations connecting to this node have an enabled gating unit.

A **CONTEXT NODE**, utilizes a custom input function to simulate input. A typical use case for this node is if we want to specify a context input to provide a top down bias at the center of the image while suppressing irrelevant locations. For example, the tWM Nodes can take in some **"PARAMS"** provided by the Cycle Controller in the vAE (see Figure C.1). This parameter specification provided by the context nodes is our input "Attentional Sample." The input function is provided by a utility class called *FcnGen* discussed in detail in Appendix A. *FcnGen* provides a useful set of prebuilt functions such as $sin(k)$ and $cos(k)$ as well as many others. After a specification of the kind of function is defined, this class can also compute its value at some point k using the call *FcnGen->compute(k)*, returning a *float*. The context node can use three independently defined *FcnGen* instances, one each for the spatial x and y and time t dimensions. This specification can be provided by the constructor *CONTEXT_NODE(FcnGen* x, FcnGen* y, FcnGen* t, int width, int height)* upon initialization and the neural activations are computed at each time step as shown by the algorithm in Listing 5.1 The *time_step* is provided by the *TN3Network* and will be discussed in a later section. Just like any of the other Nodes, the neural activation at any given time step can be retrieved using the *getOutput*() function.

```
void update(int time_step){
  for (int x; x < node.sizeX(); x++){
    for (int y; y < node.sizeY(); y++){
        return fcngen_x->compute(x)*
                 fcngen_y->compute(y)*
                 fcngen_t->compute(time_step);
    }
  }
}
```

Listing 5.1: The *update(int time_step)* function of the Context NODE class internally computes the neural activation of each neuron given the specification from the *FcnGen* class.

Figure 5.3 demonstrates the output of a context node when a Gaussian function is used for the x and y dimensions while the time dimension is kept constant. This kind of specification may be provided if the system is instructed that "the object will appear in the center of the screen" and the resulting Node can then be used as the basis for priming the VH as discussed in Section 4.3.4.

Edge (NP) Creation

The abstract **Neural Primitive** and all of the subclasses as shown in Figure 5.2 make up our NP operations. Three of our NP operations that inherit *NeuralPrimitive* represent the edges in our graph network. These operations are used to update the neural activations of Nodes by implementing the corresponding NP equation specified in Table 3.3.

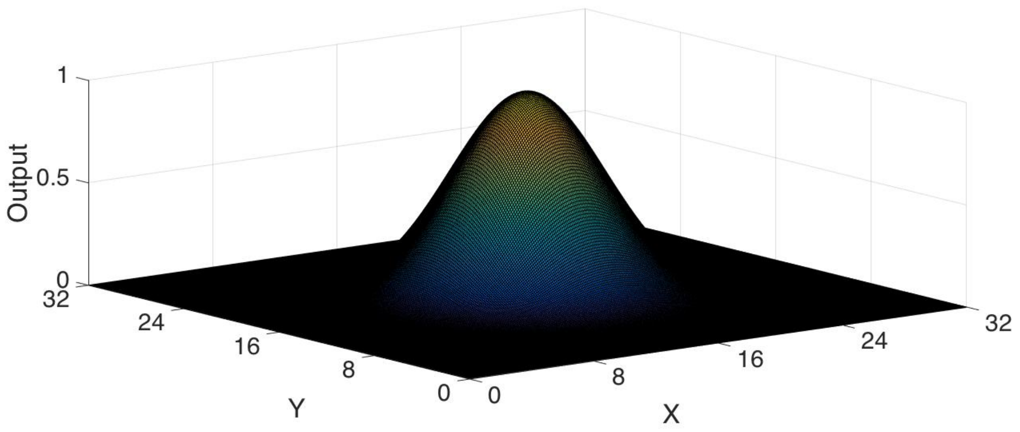


Figure 5.3: A hypothetical output of a 32×32 context node provided a Gaussian function using the *FcnGen* class in the x and y dimensions. In this example, the output does not vary with time, hence a constant input, also provided by the *FcnGen* class is used in the time t dimension.

A new instance of a *NeuralPrimitive* is created with a corresponding constructor *NeuralPrimitive(FcnGen* x, FcnGen* y, ConnectivityType type)*. The first two arguments take instances of *FcnGen*, one each for the x and y dimension for specifying the weight matrix ϕ_{AB} (see Section 3.2). Note that unlike a CNN [36] architecture where weights between neurons are automatically computed, we leave this out and manually define them. However, future work may look at automatically learning weights. The Gating NP operation is used internally by the *NeuralPrimitive* to enable and disable its computation.

The *ConnectivityType* specifies a generic connectivity pattern connecting NODE A and NODE B. This defines how two NODES may be connected together, and here we introduce two useful connectivity patterns.

1. *ONE-TO-ONE*: A single connection exists between any two neurons connecting Node A and Node B. Thus the ϕ_{AB} weight matrix is a direct mapping of every neuron from these two nodes. Requires that NODE A and B are the same size.

2. *MANY-TO-MANY*: Defines connectivity between a group of neurons from Node A converging onto some neuron in Node B similar to that used in CNN [36]. NODE A and B need not be the same size.

The newly created instance of a *NeuralPrimitive* is then added to the *TN3Network* with the function call *addOperation(NeuralPrimitive* np, Node* src, Node* dst)*. Notice here that in order to add a *NeuralPrimitive*, we must specify the two nodes that will be directionally connected together. At this point, these Nodes are assumed to already have been created as described previously.

As an example of how Nodes and Neural Primitives may be added to the network, consider

Figure 5.4 which shows a portion of the network used for priming the VH without an AS. An IT Node first needs to be created in the tWM and VH as a *Context Node* and *Default Node* respectively. Next, a *Bias Control NeuralPrimitive* operation needs to be created with a *ONE-TO-ONE* connectivity with the *FcnGen* parameters dependent on the priming specification. Finally, we create a new *TN3Network* - "PrimeWithoutAS" and the created Nodes and *NeuralPrimitive* are added. This subnetwork is then added to the **global network** as described earlier.

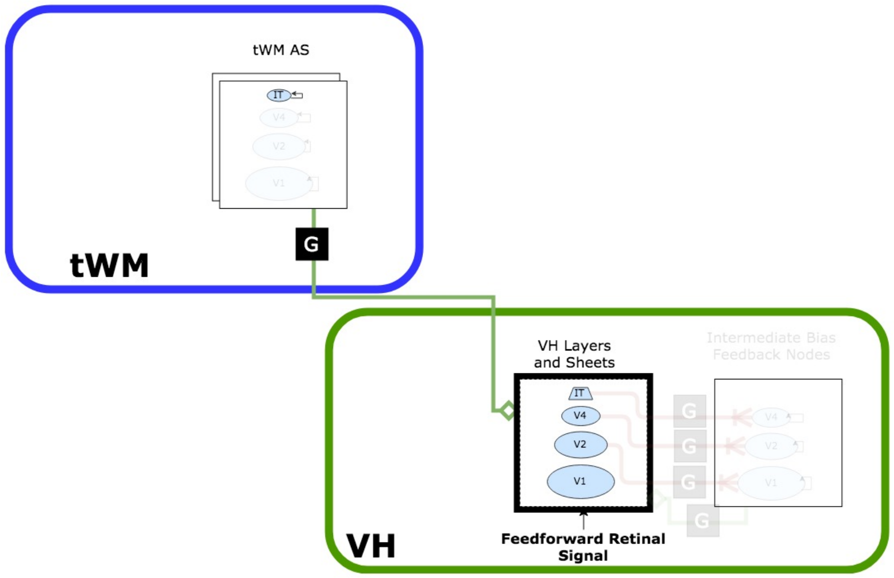


Figure 5.4: An example creation of Nodes and Neural Primitives used while priming without an AS as described in Section 4.3.4.

Graph Initialization

Once all the required Nodes and Neural Primitives are added to the *TN3Network*, it then needs to be initialized by calling the *init()* function mentioned earlier. This involves adding **intermediate nodes** that hold intermediate computations that will be performed by each NP operation of the *NeuralPrimitive* class. This is necessary since each Node in our graph that has one or more incoming connections from another Node in the graph needs to have the corresponding NP operation computed. However, without an intermediate representation, only the last operation applied on the Node would take affect, in effect ignoring all of the previous operations applied. Initializing the network also serves as a safety mechanism since it prevents any further modification to the network.

During the initialization of the *TN3Network*, there is also some error checking that must be done. For example, due to the way a *CONTEXT_NODE* is defined, it cannot be a destination node. Furthermore, a **NODE** created with a size of zero is not allowed and the *FcnGen* class must have a specification of which function to use. Errors are handled via messages like “NODE with name ’x’ in the TN3Network with name ’y’ could not be initialized. Invalid node size provided.”

Figure 5.5 shows an example of how a graph specification provided by the user consisting of two **src** nodes connecting to a **dst** node require additional **intermediate nodes** indicated by orange boxes to hold intermediate representations. Another class, called the *TN3Runner* which will be discussed in more detail in later sections is responsible for combining these intermediate representations and updating the **dst** Node.

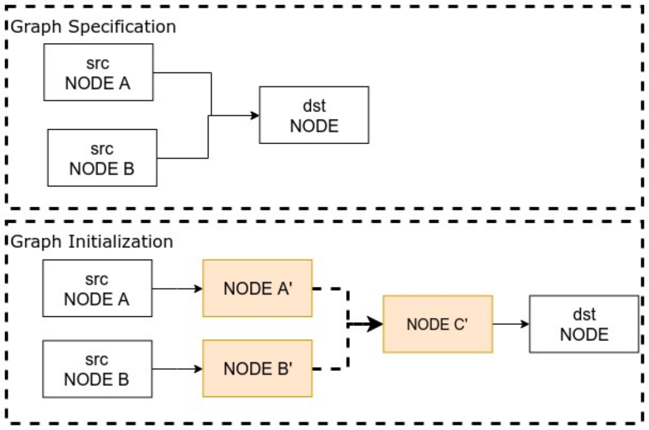


Figure 5.5: Given a graph specification, an initialized graph requires the creation of intermediate nodes (in orange) that temporarily hold computations performed by each NP operation. Solid and dotted arrows indicate the computations performed by the *NeuralPrimitive* and the *TN3Runner* class respectively.

Graph Storage

Since we can now create many *TN3Network*s, each with its own purpose such as one for priming with an AS and another for priming without an AS, networks are now modular. This allows for easier accessibility for someone that is new to TarzaNN by only considering the subnetworks they wish to modify. Furthermore, adding new networks can be easily done by adding to an existing network.

The **TN3NetworkStorage** class contains all of the stored *TN3Networks*, encapsulating Nodes and NPs that are required for all of our CPs. See the complete STAR architecture Figure C.1 in Appendix C. They can be retrieved by simple function calls that return the corresponding network. For example, to return the VH network, consisting of the predefined specification of the Nodes for layers in V1, V2, V4 and IT, we can simply call *TN3NetworkStorage::getVHNetwork()* and add this to our **global_network** using the *addNetwork()* operation mentioned earlier. See Appendix A for a complete list of stored networks.

5.2.3 Updating Nodes

A fully created *TN3Network* can now be used by another class called the *TN3Runner* to update all of the neural activations for each Node. The *update()* method in the *TN3Runner* class is responsible for executing the fundamental computations implemented by each of the Neural Primitives to transform the Nodes at each time step. The process of updating Nodes occurs by traversing through each Node in the **global_network**, including those in all of the subnetworks. For each Node, we look at the set of incoming connections and ask the corresponding NP operation to compute the **intermediate node** by calling the *compute()* function of the *NeuralPrimitive* class.

As an example, consider the priming network without an AS as discussed in Section 4.3.4. Here, the *IT* Node in the VH can have either a spatial or feature bias applied to it from nodes in the tWM, each of which is connected by a *Bias Control* NP operation. The **intermediate nodes** are first computed independently for each incoming connection and they are then combined together to form a new intermediate representation by taking the average of the Nodes (see Figure 5.5 as an example). The combined representation is then applied to the **dst** node as either feedforward input or as a multiplicative bias (see equation 5.1) depending on the specific operation used - Integration, Bias Control or Bias Feedback.

Listing 5.2 shows the pseudo algorithm to accomplish this. Notice that there are separate combined representations - *combined_inter_node_0* and *combined_inter_node_1* since the Bias Control operation applies this representation as a multiplicative bias on the **dst** node while the Bias Feedback and Integration operation applies it as feedforward input.

```
void update(){
    for each Node in global_network{
        combined_inter_node_0 = []
        combined_inter_node_1 = []
        for each incoming connection (edge / NeuralPrimitive){
            inter_node = NeuralPrimitive->compute()
            if (NeuralPrimitive->getType() == BiasControl){
                combined_inter_node_0 += inter_node
                num_bias_control_connections += 1
            } else
                combined_inter_node_1 += inter_node/num_connections
        }
        if (NeuralPrimitive->getType() == BiasControl)
            Node->setBias(combined_inter_node_0)
        else
            Node->setInput(combined_inter_node_1)
    }
}
```

Listing 5.2: Pseudo algorithm for the *update()* function in the *TN3Runner* class for updating Nodes in the global network. The intermediate nodes are first computed and then combined together. The result is applied either as feedforward input (*setInput()*) or as a multiplicative bias (*setBias()*).

5.2.4 Summary

This section highlighted how three kinds of **NODES** - Default, Memory and Context and three kinds of **NP Operations** - Integration, Bias Control, Bias Feedback could be initialized, and added to a graph in the *TN3Network* class which is then initialized.

The **TN3Runner** then traverses through each Node in the **TN3Network** and updating its neural activation given the corresponding Neural Primitive operation Nodes are connected with at each time step. In the next section, we describe how CPs can be assembled, retrieved and executed. As part of this, an important aspect will be to show how NPs can be dynamically gated given task demands.

5.3 Cognitive Programs Memory

5.3.1 Overview

The section introduces how CPs are built from *Method Elements* (CPM-ME) and then stored into the *Methods Database* (CPM-MD). Furthermore, the execution of visual tasks given a task specification provided by the Cognitive Program Compiler [32] is explained. Figure 5.6 shows a detailed view of the major components involved in these processes.

The implementation of the CPM-ME consisting of *NP operations* and *Base Methods* are present in separate classes. *NP operations* and the **NODES** on which they operate are combined in different configurations but serve a similar purpose such as priming the VH and encapsulated in the **TN3Network** as discussed in the previous section (shown as *tn3network* in Figure 5.6) . On the other hand, *Base Methods* are implemented as function calls in the **BaseMethods** class and have direct access to the **TN3Network** in order to provide functionality for *Access to Representations.*

Stored Methods in the CPM-MD are implemented in separate classes as they were introduced in Section 4. For example, the *Prime* and *Detection* methods reside in the **VH** class while the *setFHMBIas(args)* resides in the **FC** class. These classes utilize the functions in the **BaseMethods** class to compose the corresponding CPs.

The **TN3Runner** deals with all aspects of visual task execution as it interacts with several other classes. It communicates with the **vae** to provide access to methods in the CPM-MD, the **TN3Network** to update the NODES given the NP operations and the **rig** for handling changes in exogenous stimulus in the experiment and collecting neural activations for experimental data. This class is abstract and provides a template from which a task specification can be provided using the **CPC** class. It contains two abstract functions color coded in red - *init()* and *main()* that must be implemented by a concrete class inheriting from the **TN3Runner.**

The **CPC** class is part of the Cognitive Program Compiler [32] and is provided here for reference. It outputs a task script given the task specification which contains a sequence of commands including the execution of CPs after retrieval from the CPM, decision points and other functions relevant for completing a visual task. Since the CPC is beyond the scope of this thesis, we simulate its output.

In the next few sections, each of these classes just described are discussed in more detail and divided as follows.

1. *CP Assembly and Storage*: Describes how CPs in Section 4 are assembled using *Method Elements* and then stored for later retrieval.

2. *Task Specification*: Stored CPs need to be retrieved, parameterized and executed in a particular order given a visual task specification. As an example, a task specification of "press 'x' if you see a red circle " would require a *VAE()->VH()->Prime* and *VAE()->VH()->Detection* operation. This section describes how CPs can be sequenced appropriately to accomplish this.

3. *Task Execution*: Describes how a task specification is executed to update the state of the entire system until task completion criteria has been met.

5.3.2 CP Assembly and Storage

The first step in executing a visual task is to build a CP such as *VAE()->VH()->Detection* from sequencing one or more *Base Method* functions in the **BaseMethods** class with additional definition of variables, and IF-THEN-ELSE logic. CPs are stored in either the **VH** or **FC** class and are accessible through the **vae**.

The functions in this class are all static and hence do not require the class to be instantiated before being used. These functions take input directly from the parameters when the function is called. Hence, **BaseMethods** acts as a utility class where the code can be shared by multiple instances of CP methods.

Some of the *Base Methods* that we discussed in Section 4 are encapsulated as functions in the **BaseMethods** class while others such as the conditional statements are hand coded using IF-THEN-ELSE logic. The complete list of functions in this class can be found in Tables A.3,

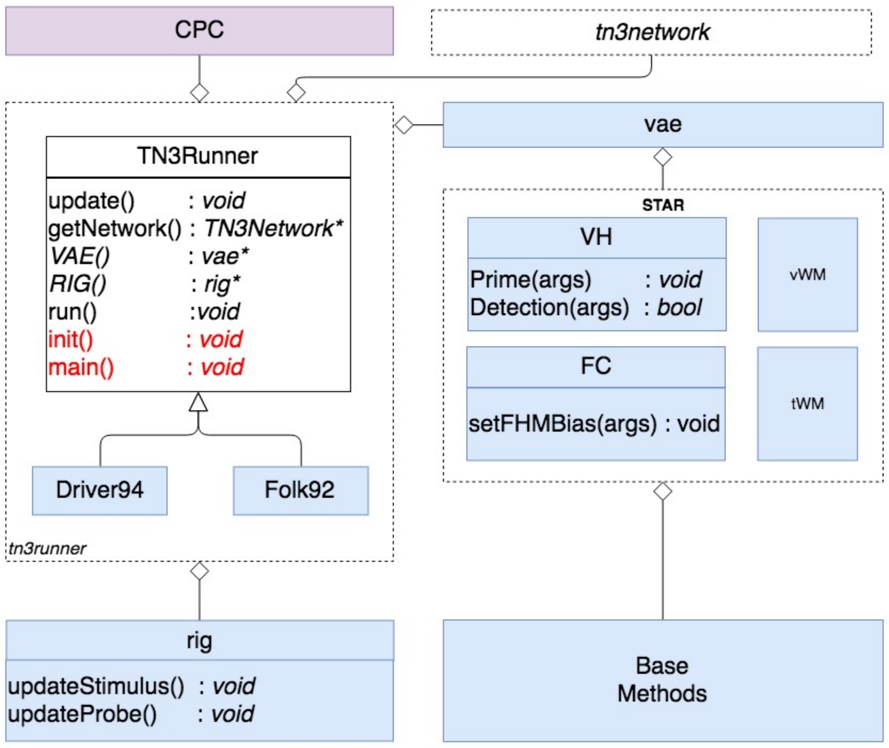


Figure 5.6: Detailed implementation of the key components in the CPM framework required for visual task execution. Red functions in the TN3Runner class are abstract. The set of functions listed for these classes is not exhaustive. See Appendix A for a complete definition of functions in each class.

A.4, A.5 and A.6 for *Access to representations*, *Parameter Specification*, *Execute a Process* and *Queries*, respectively. Below, we will discuss some of the important functions implemented in **Base Methods.**

Access to Representations involves enabling or disabling the gating of nodes and this is achieved by a call to *BaseMethod::enableGating(NeuralPrimitive* np, bool val)*. Notice that this static method takes in two arguments. First, the NP operation we wish to toggle gating on is specified followed by a boolean value indicating whether to enable (true) or disable (false) gating. Since the **TN3Network** class provides direct access to Nodes, a similar method can be used for reading and writing to a Node. Writing to a Node is useful when manually parameterizing input nodes. For example, if the tWM needs to be loaded with an AS that was previously built, we can directly set its input using a predefined method call.

Loading an attentional sample (AS) into tWM is a *Parameter Specification Base Method* which is called by *BaseMethod::loadAStoTWM(AttentionalSample* as, TN3Network* tWM_ net)*. The AS contains a vector of context Nodes each of which has a unique name indicating the layer and feature information it is storing.

When applying the AS onto the **TN3Network**, the function finds the corresponding Nodes in the tWM with the same name and applies the AS by copying the values of the neural activations from the AS to the **TN3Network**. For example, if the AS contains Nodes relevant for representing layer $V4$ of the VH, it finds the corresponding Nodes in the *tWM_net* to load the AS. If the AS is invalid such that no corresponding Nodes in the *tWM_net* are found, the function skips over the Node and produces the following warning to the user "Warning. Invalid AS specified for loading into tWM."

Base Methods involving *Execute a Process* include those for setting the relevant control signals such as disengaging attention or resetting the contents of tWM that were previously loaded. Some of these base methods are fairly involved such as the one used for the $\theta - WTA$ algorithm. However, the previous implementation of TarzaNN provided a subset of this functionality reducing the total work load required. For a definition of how these functions are called, refer to Table A.5.

Listing 5.3 shows the CP implementation for priming with an AS (see Section 4.3.3 and Figure 4.9). Lines 4, 5, 7 and 8 are used to initialize variables and retrieve the **TN3Networks**. Line 10 utilizes the **BaseMethod** class to select the top layer of the VH and return its index - an internal variable used for indexing layers within a network. Line 12 loads the entire AS supplied as input arguments into the tWM. Since the AS is a vector containing Nodes, this *Base Method* matches the AS Node name to the tWM Node name and writes the value using the *BaseMethod::writeNode(args)* described earlier. Line 14 is used to disengage any previous attention mechanisms in place. Lines 16-37 are used to iterate through each layer of the VH starting at the top *IT* layer, check if an AS exists for this layer and enable gating (line 24). In the case that a partial AS was provided as input, the priming without an AS method is called (line 33). At each iteration, the CP also calls the **TN3Runner's** update function (line 36) to update the state of the **TN3Network** at each time step as described in Section 5.2.3.

```
1  void Prime(TN3Network* attentional_sample){
2      cout << "Prime(). Begin Priming\n";
3      // Variable specification
4      string layer_name;
5      NeuralPrimitive* np;
6      // Get the Networks
7      TN3Network* VH_net = global_network->getNetwork("VH");
8      TN3Network* tWM_net = global_network->getNetwork("tWM_0");
9      // Select TOP Layer
10     int layer_indx = BaseMethod::selectLayer(VH_net, "IT");
11     // Load AS to tWM
12     BaseMethod::loadASTotWM(attentional_sample, tWM_net);
13     // Disengage Attention
14     vae->disengageAllAttentionMechanisms();
15     // Are there more layers?
16     while (layer_indx > 0){
        // Is there AS for this layer?
        if (BaseMethod::isASExist(attentional_sample,layer_indx)){
            // Specify the layer names to be gated.
            layer_name = VH_net->getNodeLayerName(layer_indx);
            np = global_network->getNetwork("PrimeWithAS")
                                  ->getOperation(layer_name);
            // Gate (AS from) tWM to VH
            BaseMethod::enableGating(np, true);
            cout << "Prime(). Working on layer (" << i
                 << ")\n";
            // Select Next Layer
            layer_indx = BaseMethod::selectNextLayer(VH_net,
               layer_name);
        }
        else{
            // Select VH.Prime W/O AS.
            cout << "Incomplete AS. Begin priming with minimum bias
               strategy."
            vae->VH()->Prime();
            break;
        }
        runner->update(LAYER_DELAY);
    }
    cout << "Prime(). Finished Priming\n";
}
```

Listing 5.3: Assembling of the CP for priming with an attentional sample requires the use of several Base Method functions as well as boolean logic.

5.3.3 Generate TN3Runner Task Specification

Once all of the required CPs have been assembled and stored into the **VH** and **FC** class as described above, a visual task may be executed using these CPs. In order to execute a visual task, a task specification needs to be provided that indicates which CPs to use, their order of execution and their parameters. Kunic's CPC [32] provides the basis for this specification. However, since work is still underway to harmonize the implementation of the CPC with the CPM framework described here, we opted for an alternative method for generating the task specification.

Task script specification is the starting point for visual task execution and it begins with a visual task specification provided by the **CPC** class. Each task script is different depending on the visual experiment but shares some similarities. A task script is a subclass of **TN3Runner** and is composed of code blocks relevant for initialization and CP execution.

The **TN3Runner** shown in Figure 5.6 is abstract while the **Driver94** and **Folk92** are two example concrete classes with a task specification. Code relevant to all aspects of visual task execution is implemented here. In order to implement a concrete class, the two functions shown in red in Figure 5.6 - *init()* and *main()* must be implemented.

Code blocks for initialization (*init()*) include information about the sequence in which stimuli will be presented to the system, their duration and the directory they are located in. It also specifies useful run time parameters such as for $\theta - WTA$ and the duration of $L - IOR$. This code block may also include tweaking the filter parameters for the VH to optimize feature selectivity or indicating the number of tWM **TN3Networks** to create for holding multiple AS's. Listing 5.4 shows a sample initialization function divided into three aspects. Line 3 calls the super constructor to initialize the **VAE, RIG** and the **TN3Runner** *global_ network*. The **VAE** class is responsible for communicating with all major components of STAR; the VH, FC, vWM and tWM. This includes setting any relevant control signals and parameterizations. The **RIG** class is used for recording neural activations of either all the neurons within a Node or a subset - useful for collecting experimental data for analysis. It is also used for displaying stimulus to the system and specifying its duration.

Line 5 is used to retrieve and initialize all of the stored **TN3Networks** and then add them to the global network (line 8). Once all of the networks are added, the global network is initialized (line 11). In the experiments that will be presented in Section 6, this function initializes the complete network shown in Figure C.1 (see Appendix C). Parameter specifications are then provided (line 13) followed by a definition of all stimuli that will be presented to the system (line 16). The *addStimulus(...)* function takes in as argument the name of the image file that will be used as stimulus onset, the internal name of the stimulus and an integer specifying its duration. During visual task execution, the relevant stimulus is automatically provided to the system.

```
1  void init(){
2      /* --- Call the superclass constructor --- */
3      TN3Runner::init();
4      // Get TN3Networks
5      TN3Network* vh_net = TN3NetworkStorage::getVHNetwork();
6      ...
7      // Add TN3Networks to global_network
8      global_network.add(vh_net);
9      ...
10     // Initialize the TN3Network
11     global_network.init()
12     /*  --- Parameter Specification Code Block --- */
13     VAE()->setWTATheta(20.0);
14     ...
15     /* --- Stimulus Onset Specification --- */
16     RIG()->addStimulus("inter-trial.png","inter-trial",500);
17     ...
18 }
```

Listing 5.4: A sample initialization function *init()* includes all the relevant code for setting up the **TN3Network** and task dependent parameter specification before visual task execution. For a full list of parameter specification see Implementation.

The second code block is the *main()* function which includes all the relevant commands for CP execution. User specified commands such as "press 'x' if you see a red circle " are converted into function calls (i.e. *VAE()->VH()->Prime({'color':'red'})*) by the **CPC** class. Listing 5.5 shows the implementation of this code block for one of the experiments that will be discussed in more detail in a subsequent section. The structure typically follows some variable initialization (lines 3, 4 and 6) followed by visual task execution (lines 8-37).

```
1  void main(){
2      // Variable Initialization
3      bool detection;
4      AttentionalSample* prime_as;
5      // Retrieve the stored AS for target detection
6      AttentionalSample* square_as = StoredAS::getSquareAS();
7      // Display inter-trial image
8      VAE()->waitForStimulus("Inter-trial");
9      // Look at the fixation cross
10     VAE()->FC()->setFHMBias({"location":"centre"},0.8);
11     // A fixation cross with two rectangles will appear.
12     VAE()->waitForStimulus("Fixation");
13     // A cue will appear for 100 ms then disappear.
14     VAE()->waitForStimulus("Cue Onset");
15     // Get Attentional Sample
16     prime_as = VAE()->VH()->getAS();
17     // Transform AS to represent location information only.
18     transformAStoLocationAS(prime_as);
19     // Prime the VH
20     VAE()->VH()->Prime(prime_as);
21     // Load the AS to tWM
22     BaseMethod::loadAStoTWM(square_as,
23                             global_network->getNetwork("tWM_net_0"));
24
25     // A target will appear on the screen for < 2,000 ms.
26     VAE()->waitForStimulus("Target Onset");
27     // When you see the square, respond.
28     detection = VAE()->VH()->Detection(FEATURE_DETECTION);
29     if (detection == true)
30        cout << "Target detected. Task Complete";
31     else{
32        cout << "Target not found. Lifting all attention mechanisms";
33        // Disengage all attention mechanisms.
34        VAE()->disengageAllAttentionMechanisms();
35        // Try to find the target again.
36        detection = VAE()->VH()->Detection(FEATURE_DETECTION);
37        if (detection)
38            cout << "Target detected. Task Complete";
39        else
40            cout << "Target not detected. Task Complete";
41     }
42 }
```

Listing 5.5: The sequence of commands required for executing the Egly and Driver '94 experiment by the TN3Runner class (See Section 6)

Now that a description has been provided of the constituents of a task script including how all of the relevant *TarzaNN* system parameters are initialized and the commands called for CP execution, we look at how CP methods are assembled and executed.

5.3.4 Task Execution

Once a **TN3Runner** subclass with task dependent specification has been created, a visual task may now be executed by calling **TN3Runner's** *run()* method. This function initializes the system by calling the *init()* method and executes the relevant task script by calling the *main()* method as shown in Listing 5.6. This function is already implemented in the abstract **TN3Runner** class and does not need to be specified again by the user.

```
void run(){
    // Initialization before visual task execution.
    init();
    // Task specification and visual task execution.
    main();
}
```

Listing 5.6: Task execution by the **TN3Runner** occurs by calling the *run()* method.

5.3.5 Summary

In this section we introduced how Cognitive Programs (CPs) can be assembled using methods in the **BaseMethods** class as well as additional logic where necessary. Assembled CPs are accessible through either the **VH** or **FC** class depending on the recognition method they belong to.

Given these CPs, we then described how a task specification can be used to implement a concrete **TN3Runner** class where the user must specify what needs initialization (*init()*) and what needs to be executed for a given visual task (*main()*). While the initialization method deals with setting up parameters, **TN3Networks** and stimulus onsets and durations, the main method deals with the ordering in which CPs are executed and which parameters they require.

Finally, a visual task may be executed by calling the **TN3Runner's** *run()* method. During execution, the state of the system including variables **and TN3Networks** are updated at each time step until the task termination criterion has been met.

6 Evaluation

This section contains visual attention and neural tuning curve experiments which demonstrate several CPM-M's within the CPM framework.

6.1 Visual Attention Experiments

The goal of these visual attention experiments is to demonstrate a variety of the CPs functionality in our CPM. Here we show that the task dependent execution of these CPs results in qualitatively similar reaction times and correct task decision output with respect to three psychophysical experiments: a) Egly & Driver '94 [19], b) Folk et al. '92 [21], and c) Raymond et al. '92 [57].

The reaction times obtained by the system have the same relative ordering of the reaction times across the different test conditions in our experiments. Since the CPs are a proof of concept at this point, only a qualitative comparison of the reaction times is provided.

For each of these visual experiments, a $C++$ version of the set of CPs the vTE executes are shown. This code is a modified version of the commands executed by the output of Kunic's CPC [32], and although it isn't a one-to-one reproduction, the major CPs executed remain the same.

6.1.1 Egly & Driver '94

In Egly & Driver '94, the authors looked at how shifts in visual attention between objects and locations were reflected in the behavioral performance of subjects using a spatial cue [19]. They found that there was an advantage in reaction time when the target appeared on the same object versus a different object in an invalid cue condition indicating an object based attention component.

However, in our demonstration of some key Cognitive Programs - those involved for spatial priming and detection, we focus only on the effects of spatial attention. In this regard, the authors found that spatial attention resulted in behavioral performance on a validly cued trial compared to the invalidly cued trial irrespective of which object the target appeared.

Two rectangles and a fixation cross were presented and the subject is cued by a whitening of any of the four corners of the rectangle. After the presentation of the inter-stimulus interval (ISI) a target is presented at either the cued or any of the uncued locations (see Figure 6.1). The user is asked to respond by pressing a button if they have detected the target "as rapidly as possible" while maintaining fixation at the centre of the display in a covert attention only paradigm. The target remains on the screen for up to 2000 ms or until the user responds.

The results of the experiment showed that spatial attention negatively effected behavioral performance when an invalid cue was presented (mean 364 ms with target appearing at same object different location and different object uncued location), resulting in a reaction time cost of 40 ms higher than in the valid cue condition (324 ms) [19].

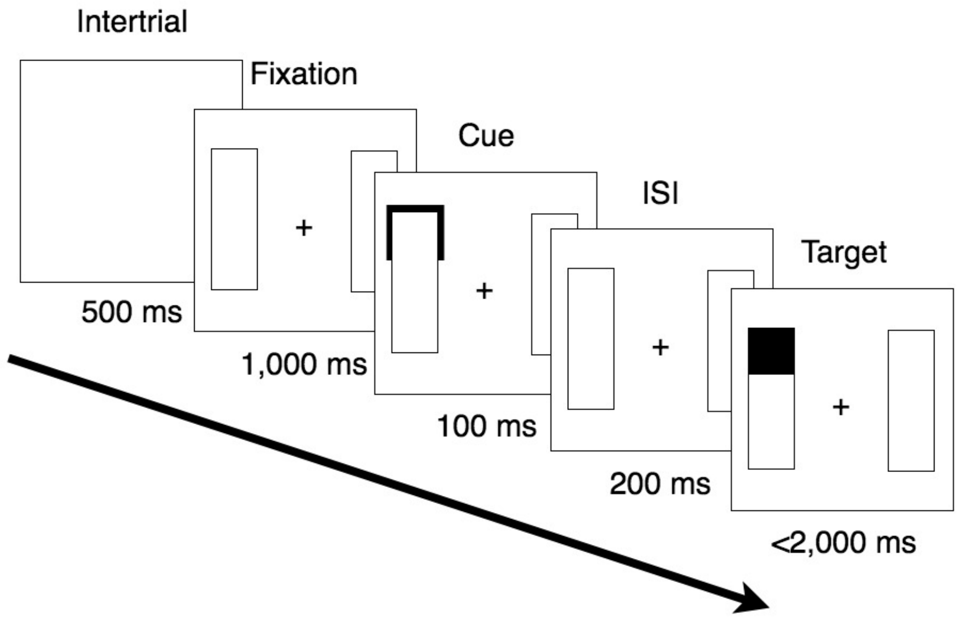


Figure 6.1: Experiment sequence for Egly & Driver 1994 on a correctly cued trial. Reproduced from [19].

vTE CP Execution Listing 6.1 shows the CPs executed by the vTE during this visual task.

```
void main(){
    // Variable Initialization
    bool detection;
    AttentionalSample* prime_as;
    // Retrieve the stored AS for target detection
    AttentionalSample* square_as = StoredAS::getSquareAS();
    // Display inter-trial image
    VAE()->waitForStimulus("Inter-trial");
    // Look at the fixation cross
    VAE()->FC()->setFHMBias({"location":"centre"},0.8);
    // A fixation cross with two rectangles will appear.
    VAE()->waitForStimulus("Fixation");
    // A cue will appear for 100 ms then disappear.
    VAE()->waitForStimulus("Cue Onset");
    // Get Attentional Sample
    prime_as = VAE()->VH()->getAS();
    // Transform AS to represent location information only.
    transformAStoLocationAS(prime_as);
    // Prime the VH
    VAE()->VH()->Prime(prime_as);
    // Load the AS to tWM
    BaseMethod::loadAStoTWM(square_as,
                            global_network->getNetwork("tWM_net_0"));

    // A target will appear on the screen for < 2,000 ms.
    VAE()->waitForStimulus("Target Onset");
    // When you see the square, respond.
    detection = VAE()->VH()->Detection(FEATURE_DETECTION);
    if (detection == true)
       cout << "Target detected. Task Complete";
    else{
       cout << "Target not found. Lifting all attention mechanisms";
       // Disengage all attention mechanisms.
       VAE()->disengageAllAttentionMechanisms();
       // Try to find the target again.
       detection = VAE()->VH()->Detection(FEATURE_DETECTION);
       if (detection)
           cout << "Target detected. Task Complete";
       else
           cout << "Target not detected. Task Complete";
    }
}
```

Listing 6.1: The sequence of Cognitive Programs executed by the vTE.

Initialization Before the experiment begins, the system must have been told the range of output activations that constitute a positive match of the target, specified by the decision criteria. Note here, that we assume that the attentional sample of the target has already been built and stored in tWM, here on referred to as *square_as*. The minimum and maximum detection values in the function below are heuristically chosen.

```
VAE()->VH()->setDetectionCriteria(0.1,0.26);
```

Building an Attentional Sample The cue is presented to the system by the vTE at $t = 0$ ms. At a later time, $t = 90$ ms a selection is made at IT and the system now begins to build an attentional sample. Recall that this process can be terminated earlier if there is not enough time to fully traverse down the VH. Our system stops after layer IT because a selection is no longer made in lower layers due to the removal of the cue stimulus. Once this process returns, the attentional sample is transformed to represent location information only and abstract away any feature information. See Section 4.3.2 for a discussion.

```
prime_as = VAE()->VH()->getAS();
transformASToLocationAS(prime_as);
```

Priming the VH After an attentional sample has been built and transformed, the system is ready to prime the visual hierarchy. The process begins upon ISI onset and continues until it is interrupted by a change of visual stimulus or the process is complete down all layers of the visual hierarchy. The process works by gating the contents of the AS, now stored in tWM to the VH.

```
VAE()->VH()->Prime(prime_as);
```

Target Detection Target detection begins when the target stimulus is presented at $t = 300$ ms after cue onset. First, a selection is made at the very top of the VH, then these contents are gated to vWM. The vWM is then compared against the attentional sample in tWM and a decision is made given the detection criteria.

```
result = VAE()->VH()->Detection(FEATURE_DETECTION);
```

Decision If the target is successfully detected, then the CP execution terminates indicating the time the target was detected. If the process fails, then the system disengages attention and attempts to locate a new winner.

```
VAE()->disenageAllAttentionMechanisms();
```

After attention is disengaged, the detection process mentioned previously is initiated for a second time. If the system still fails to find the target, the program terminates indicating that there was no target found.

Results Table 6.1 shows the selection, priming and target detection times at each layer of the visual hierarchy. Note that some layers are greyed out indicating that the process did not occur. In the cue presentation, there was insufficient time for a feedback pass due to the limited presentation time, thus only information at the top most IT layers are available. In the target detection, the cells are greyed out since the information available at the top of the VH was sufficient for the matching process. Note that a minimum bias strategy is employed at layers V4, V2, V1.

This experiment thus demonstrates the functionality of three CPs: *VH.Detection*, *VH.Localization* and *VH.Prime (with AS)*.

Layer	**Cue** Selection Time (ms)	**ISI** Priming Times (ms)	**Valid Target** Target Detection Time (ms)	**Invalid Target** Target Detection Time (ms)
IT	90	110	390	420
V4	▨	120	▨	▨
V2	▨	130	▨	▨
V1	▨	140	▨	▨

Table 6.1: Cue detection, priming and target detection times for each process and the current stimulus presented to the system. Greyed out boxes indicate that the process was not initiated at this layer of the VH.

Figure 6.2 qualitatively compares the reaction times found in Table 6.1 to those in the original psychophysical experiment indicating that the system is able to reproduce the authors original findings. Note the limitation of this comparison to the original experiment is that the invalidly cued target RTs are all collapsed together with the mean taken of the target presented at the same object different location and different object uncued location since the aim was to show the usage of key Cognitive Programs - those involved in priming and detection. Future work may look at testing the object based attentional component differences in RT in this experiment.

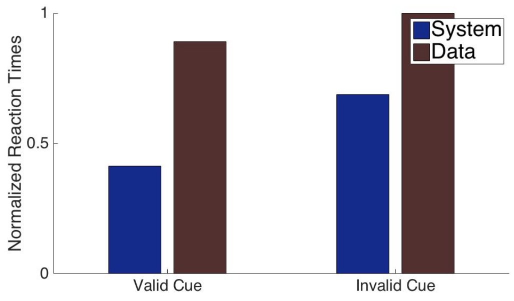


Figure 6.2: Qualitative comparison of the reaction times for target detection in Experiment 1 of [19] and in our simulation results. The normalized RT in the invalid cue is the mean of the target presented at the same object different location and different object uncued location.

Listing 6.2 shows the terminal output of the experiment on an invalidly cued trial. The numbers on the left hand side indicate the time (in ms) the command was initiated. Notice that the system attempts to detect the target at t = 1850 ms but fails because the detection criterion is not met.

```
0     AE::updateStimulus(). Presenting Stimulus (Intertrial)
50    FC::setFHMBias({"location": "center"}).
500   AE::updateStimulus(). Presenting Stimulus (Fixation)
1500  AE::updateStimulus(). Presenting Stimulus (Cue)
1570  AE Selection signals 1,0,0,0,0
1570  VH::getAS(). Begin retrieving attentional sample.

1600  AE::updateStimulus(). Presenting Stimulus (Fixation)
1600  VH::Prime(). Begin Priming
1610  VH::Prime(). Working on layer 4
1620  VH::Prime(). Working on layer 3
```

```
1620  VH::Prime(). Attentional Sample Not Found. Using minimum bias.
1630  VH::Prime(). Working on layer 2
1640  VH::Prime(). Working on layer 1
1640  VH::Prime(). Finished Priming

1800  AE::updateStimulus(). Presenting Stimulus (Target)

1800  VH::Detection(). Begin Detection
1840  AE Selection signals 1,0,0,0,0
1850  VH::getMatchDetectionResult: 0.03
1850  Target not found. Lifting all attention mechanisms.

1860  VH::disengageAttention(). Working on layer 5
1870  VH::disengageAttention(). Working on layer 4
1880  VH::disengageAttention(). Working on layer 3
1890  VH::disengageAttention(). Working on layer 2
1900  VH::disengageAttention(). Working on layer 1


1900  VH::Detection(). Begin Detection
1940  AE Selection signals 1,0,0,0,0
1950  VH::getMatchDetectionResult: 0.12
1950  Target detected. Task Complete.
```

Listing 6.2: Terminal output of the experiment in an invalidly cued trial.

6.1.2 Folk et al. '92

The Folk et al. 1992 experiment [21] is a recognition pop-out search task requiring covert attention only. Four square boxes are presented peripherally and one square box centrally. A cue indicated by four red filled circles around one of the square boxes indicates the expected location the target will appear. The uncued boxes were all surrounded by white unfilled circles. Here, the subject is told before the experiment begins whether this trial will have a valid or invalid cue. Hence, the subject must use this information when discriminating the target. A fixation display is presented after the color cue followed by a color target. The target is a solid 'X' or '=' and the distractors are non-solid stimuli all shown in the peripheral squares. The central square never has a target present. The user is asked to discriminate the target (was it an 'X' or '='?) and respond accordingly as fast as possible. The goal of this experiment is to investigate how reaction times are affected by the interaction between the cue and the properties required for task performance due to involuntary shifts of covert attention. The authors found that costs only occur during the presentation of an invalid cue that represents the same property (for example, color or location) as the target, in this condition there was an increase in reaction times.

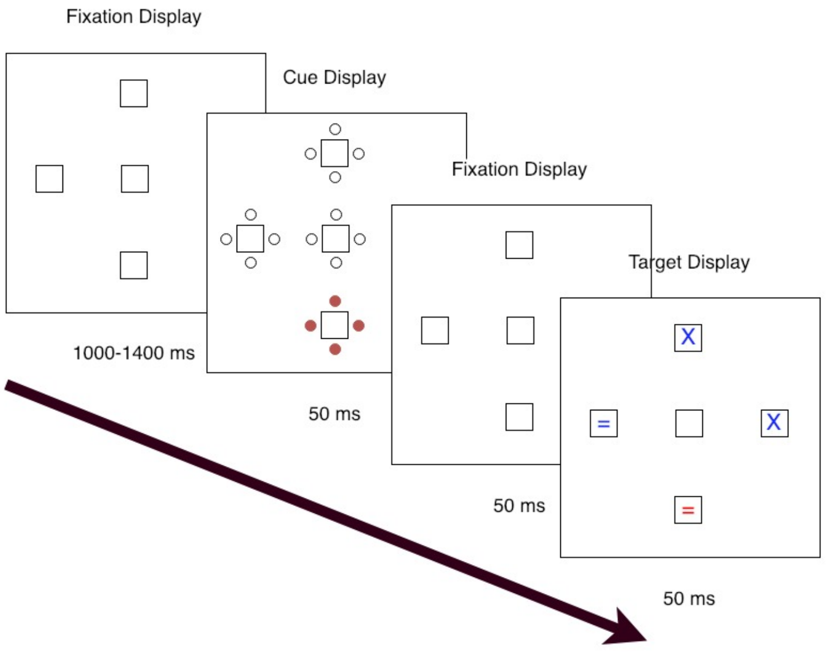


Figure 6.3: Experiment sequence for Folk et al. 1992 on a correctly cued trial. Reproduced from [21].

vTE CP Execution The sequence of CPs executed by the vTE during the visual task is shown in Listing 6.3.

```
void main() {
    // Variable Initialization
    int result;
    AttentionalSample* prime_as;
    AttentionalSample* equal_as = StoredAS::getEqualsAS();
    AttentionalSample* x_as = StoredAS::getLetterXAS();
    // Display fixation image
    VAE()->waitForStimulus("Fixation");
    // Look at the center fixation box.
    VAE()->FC()->setFHMBias({"location":"centre"},0.8);

    // A cue will appear for 50 ms then disappear.
    VAE()->waitForStimulus("Cue");

    prime_as = VAE()->VH()->getAS();
    VAE()->VH()->Prime(prime_as,{'color':'red'});

    // Load AS into tWM
    BaseMethod::loadAStoTWM(equal_as,
                            global_network->getNetwork("tWM_net_0"));
    BaseMethod::loadAStoTWM(x_as,
                            global_network->getNetwork("tWM_net_1"));

    // A target will appear for 50 ms then disappear.
    VAE()->waitForStimulus("Target");

    // Press '0' or '1' if you see a '=' or 'X' respectively.
    result = VAE()->VH()->Recognition(FEATURE_DETECTION);
    if (result == 1)
        cout << "Target discriminated. 'X' found. Task Complete";
    if (result == 0)
        cout << "Target discriminated. '=' found. Task Complete";
}
```

Listing 6.3: The sequence of Cognitive Programs executed by the vTE.

Initialization Where as the $VH.Detection$ decision criterion in the previous experiment takes in two numbers as arguments - the minimum and maximum threshold values, $VH.Recognition$ decision criterion requires a vector of minimum and maximum threshold values. For this experiment, the decision criterion is provided with two elements in the vector to indicate how well the contents of the tWM match the incoming stimulus. The contents of tWM contain two templates for the targets 'X' and '=', from here on referred to as *equals_ as* and *x_ as* as partial attentional samples**.** To ensure there is no ambiguity, the discrimination task returns the integer of the index of the template that matches if and only if it meets the search criterion defined below.

```
VAE()->VH()->setRecognitionCriteria({0.2,0.4},
                                     {0.2,0.4});
```

Building an Attentional Sample After presentation of the cue at $t = 1500$, a winner is selected at the top most layer IT. In the valid cue condition, the system knows that the cue is 100% indicative of the location the target will appear. Likewise, the invalid cue condition indicates that the target will never appear at this location. However, since these control signals are assumed to be involuntary as suggested by the authors, the same priming effect occurs.

Once the cue is detected by the system, it is almost immediately removed due to its short presentation duration of 50 ms. Hence, only course spatial location information is extracted at IT . Note that the inferotemporal (IT) neurons can have receptive fields as large as 23° in diameter from the centre of gaze [26], and, hence priming will be spatially course down the VH. The attentional sample is then transformed to represent location information only abstracting away feature information.

```
prime_as = VAE()->VH()->getAS();
transformASToLocationAS(prime_as);
```

Priming the VH The visual hierarchy is primed using the partial attentional sample and the minimum bias strategy is employed at layers V4, V2 and V1. Task knowledge indicating that we are looking for a red target is applied at the same time.

```
VAE()->VH()->Prime(prime_as,{"color":"red"})
```

Loading AS into tWM Prebuilt AS's representing the two possible targets - an 'X' and a '=' are loaded into separate *tWM Networks*. The networks are retrieved from the *global_network* by its name - *tWM_net_1* and *tWM_net_2*.

```
BaseMethod::loadAStoTWM(equal_as,
                        global_network->getNetwork("tWM_net_1"));
BaseMethod::loadAStoTWM(x_as,
                        global_network->getNetwork("tWM_net_2"));
```

Target Recognition Upon target presentation, we assume that the target is salient and detected via a 'pop-out' bottom up process since it was primed for earlier and since it automatically captures bottom up attention due to its distinct features with respect to distractors [80].

```
result = VAE()->VH()->Recognition(FEATURE_DETECTION);
```

Results Table 6.2 shows the selection and priming times during the cue and ISI stimulus respectively.

Layer	**Cue Presentation** Selection Time (ms)	**ISI Presentation** Priming Times (ms)
IT	50 (0)	60
V4	▨	70
V2	▨	80
V1	▨	90

Table 6.2: Cue detection and priming times.

Table 6.3 shows the selection and recognition times of the target after cue onset. In addition, the output of the system is also shown by the *Discriminated Target*. The similarity vector shown in the last column indicates how well the contents in the vWM match those of the tWM. Here, higher values indicate a better match. Note how the similarity values are lower for the invalid cue condition

Cue Condition	**Ground Truth Target**	**Discriminated Target**	**Recognition Time (ms)**	**Similarity Vector**
100% Valid Cue	X (Valid Cue)	X	240	0.32
	= (Valid Cue)	=	242.5	0.24
100% Invalid Cue	X (Invalid Cue)	X	285	0.18
	= (Invalid Cue)	=	287.5	0.10

Table 6.3: Target selection and recognition times after cue onset. The similarity vector shows how well the detected target matches the template target stored in the tWM. Higher values indicate a closer match.

Figure 6.4 shows the reaction times the system outputs in the valid and invalidly cued condition plotted against the findings in the original experiment.

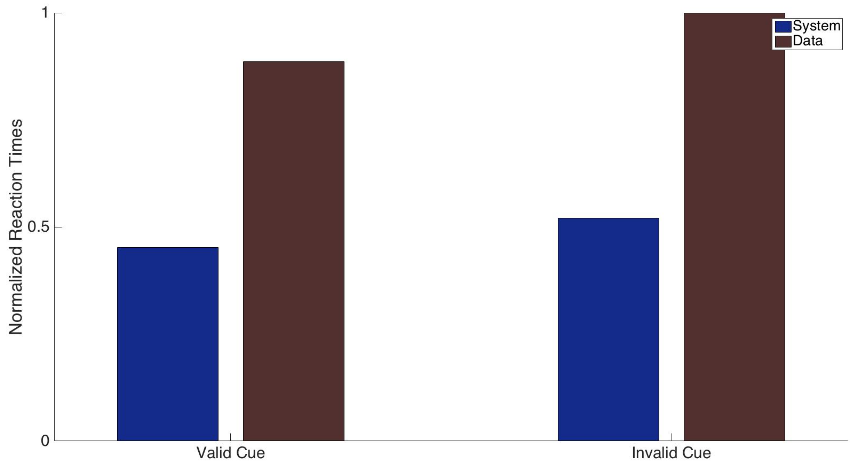


Figure 6.4: Qualitative comparison of the reaction times outputted by the system to those found in the original Folk et al. 1992 experiment [21].

Listing 6.4 shows the terminal output during the execution of the task.

```
0     AE::updateStimulus(). Presenting Stimulus (Fixation)
50    FC::setFHMBias({"location": "center"}).
1000  AE::updateStimulus(). Presenting Stimulus (Cue)

1050  AE::updateStimulus(). Presenting Stimulus (Fixation)

1060  AE Selection signals 1,0,0,0,0
1060  VH::getAS(). Begin retrieving attentional sample.


1050  VH::Prime(). Begin Priming
1050  VH::Prime(). Working on layer 4
1060  VH::Prime(). Working on layer 3
1060  VH::Prime(). Attentional Sample Not Found. Using minimum bias.
1070  VH::Prime(). Working on layer 2
1080  VH::Prime(). Working on layer 1
1090  VH::Prime(). Finished Priming

1100  AE::updateStimulus(). Presenting Stimulus (Target)

1160  VH::Recognition(). Begin Discrimination
```

```
1160  AE Selection signals 1,0,0,0,0
1250  VH::getMatchDetectionResult(0): 0.03
1250  VH::getMatchDetectionResult(1): 0.24
1250  Target discriminated. 'X' found. Task Complete.
```

Listing 6.4: Terminal output of the experiment in an validly cued trial.

Summary This experiment shows that involuntary shifts in attention can increase the reaction times for recognition of the target if the cue and target share similar properties. Here, we showed that cueing was processed in exactly the same way irrespective of whether the target was incorrectly or correctly cued. However, the key finding here was that since there was more noise on an incorrectly cued trial, the recognition process took longer to reach a threshold in order to reach a decision, reflected in the reaction times between the tested conditions.

6.1.3 Raymond et al. '92

In the Raymond et al. 1992 [57] experiment, subjects were presented with a series of letters in a rapid serial visual presentation (RSVP) paradigm. Within the stream of letters the subject was asked to identify the white target and detect a partially specified probe appearing some time after target presentation.

From here on in, a probe presented immediately after the target is denoted as (+1), while a probe appearing one stimuli after the target is denoted as (+2) and so on. The authors found that the probability of detection of the probe increased if it was presented immediately after the target or much later in the visual stream with respect to the target. However, during an intermediate duration there appeared to be a temporary suppression mechanism in place suggested by a significantly decreased probability of detection.

The source of this temporary deficit is thought to be due to target identification processes that interfere with probe detection since the effect was not seen when a brief blank interval followed target identification. In our paradigm, all stimuli including the probe were green in color while the white target was modeled as a red color. See Appendix C for more details about stimulus parameterization.

In the work by Sengupta et al. [70], this temporary deficit in probe detection was hypothesized to be due to a top down inhibitory signal to the visual hierarchy which is initiated immediately following target feature detection during the memory consolidation phase. In other words, after a target has been detected the system is in the state "I have found a white target, but I do not know what letter the target is." In order to ensure accurate target identification, subsequent stimuli in the VH stream need to be suppressed from reaching the vWM. During this period of time, subsequent incoming stimuli have a reduced probability of being detected.

At the current time, CPs are deterministic,. Because of this, we will not be able to directly show the probability of detection and identification of the probe and target, respectively. Instead, we show that the similarity between the template and incoming stimuli stored in vWM directly correlate with the probability of detection of the probe.

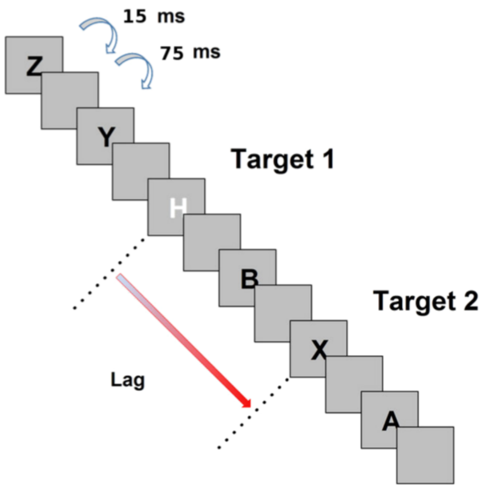


Figure 6.5: Raymond et al. 1992 experiment paradigm.

vTE CP Execution The sequence of CPs executed by the vTE during the visual task is shown in Listing 6.5.

```
void main() {
    bool TARGET_FOUND = false;
    int  TARGET_IDENTITY = -1;
    bool PROBE_FOUND  = false;
    // Display fixation image
    VAE()->waitForStimulus("Fixation");
    // Look at the center fixation box.
    VAE->FC->setFHMBias({"location":"centre"},0.8);
    // Load AS's into tWM
    for (int i = 0; i < NUM_RAYMOND_AS; i++){
        AttentionalSample* as = StoredAS::getRaymondAS(i);
        BaseMethod::loadAStoTWM(as,
                                global_network->
                                getNetwork("tWM_net_" + to_string(i))
                                    );
    }

    // A sequence of letters will appear.
    for (int i = 0; i < NUM_IMAGES; i++){
       VAE()->waitForStimulus(letter[i]);
       if (!TARGET_FOUND){
          TARGET_FOUND = VAE()->VH()->Detection({"color":"red"});
          TARGET_IDENTITY = VAE()->VH()->
                            Identification(OBJECT_DETECTION);
          cout << "Target found.";
          continue;
        }
        if (!PROBE_FOUND && TARGET_FOUND){
            PROBE_FOUND = VAE()->VH()->Detection(letter_x_template);
            cout << "Probe Found.";
        }
    }
    cout << "Task complete.";
}
```

Listing 6.5: The sequence of Cognitive Programs executed by the vTE.

Initialization Before running the experiment, templates of all letters used in the experiment were created as partial attentional samples and stored into tWM. In the code listing above, this is referred to as a vector of attentional samples: *letter_templates.* The identification criteria indicates the range the output neural activation should be in of the chosen class in order to correctly identify the target. If the neural activations do not fall within this threshold, then the system returns -1 indicating that it has not been detected.

```
VAE()->VH()->setIdentificationCriteria(args);
```

Next, the strength and duration of the memory consolidation signal must also be specified. This signal is initiated by the execution of the *VH.Identification* process immediately after target detection and modeled as a multiplicative top down bias using the Bias Control operation. This aims to suppress any incoming stimuli while the target is in the process of being stored into vWM. See Appendix B for an effect of this on neural tuning curves in area IT. The function takes in as argument the initial maximum strength (1.0) and the length of the signal (500) in ms. The values are heuristically chosen here to best represent the duration the interference occurred in the original experiment.

```
VAE()->VH()->setMemConsolidationParams(1.0, 500.0);
```

Target Detection During the RSVP, the system must continuously monitor to see if the target has been detected. Since we employ detection twice in this experiment (once during detection of the target and once during detection of the probe), we may specify the criteria for each just before calling the *VH.Detection* CP.

```
VAE()->VH()->setDetectionCriteria(0.25,0.4);
result = VAE()->VH()->Detection({"color":"red"});
```

Target Recognition If a target has been detected, the system attempts to identify the target. Note that this process also initiates the memory consolidation signal at the top of the VH.

```
result = VAE()->VH()->Identification(OBJECT_DETECTION);
```

Probe Detection After the above process is complete, the memory consolidation signal is still engaged at IT. Even so, the system continues to proceed and attempts to detect the probe. We set the detection criteria as we did for target detection.

```
runner->VAE()->VH()->setDetectionCriteria(0.09,0.40);
result = VAE()->VH()->Detection(OBJECT_DETECTION);
```

Results The terminal output has been omitted due its large length. Instead, we provide evaluation of the detection and identification times. The first portion of this experiment deals with correctly detecting and identifying the target and the results are shown in Table 6.4. Notice that detection precedes identification as reflected in the times the respective process finished executing.

Ground Truth Target	Detected Target	Detection Time (ms)	Identification Time (ms)	Similarity Vector
A	A	60	73.5	0.33
E	E	62.5	75	0.28
X	X	62.5	75	0.39

Table 6.4: Target detection and identification times in ms from time of stimulus onset. The similarity vector indicates how well the target represents the template stored in tWM. Higher values are better.

The next stage revolves around probe detection. After target detection, a top down memory consolidation signal is initiated at the top most layer of the VH. When the signal reduces in intensity, matching the incoming stimuli becomes more reliable as reflected in the similarity vector shown in Table 6.5.

The last row of the table indicates the result of probe detection when it is presented anytime after target onset, but no target identification processes have been initiated, serving as a control. Here, the values are averaged when the probe is presented +1, +2, +3 and +4 after target onset. Notice that the similarity value is significantly higher than when target identification processes are initiated. This is in line with the authors prediction that the probability of probe detection is negatively effected only when the user is instructed to perform target identification [57].

Detected Probe	Presentation Relative to Target	Detection Time (ms)	Similarity
X	+1[180 ms]	60.0	0.09
X	+2 [270 ms]	62.5	0.12
X	+3 [360 ms]	62.5	0.13
X	+4 [450 ms]	62.5	0.16
X	+7 [450 ms]	60.0	0.35
X	Control, no identification.	60.0	0.35

Table 6.5: Probe detection times in ms from time of stimulus onset. The similarity value indicates how well the target represents the template stored in tWM. Higher values indicate a better match.

To evaluate our results to those found by the authors, we look at Figure 3A in [57]. Here, the authors plot the % correct probe detection as a function of the relative serial position of the probe. Figure 6.6 shows a plot of the experimental findings reproduced from [57] reported as 'Data.' The values reported for the 'System' are those found in Table 6.5. Note that we were not able to replicate the findings in the (+1) and (+2) conditions due to the memory consolidation signal taking place immediately after target identification. Additional mechanisms may be needed to account for this anomaly.

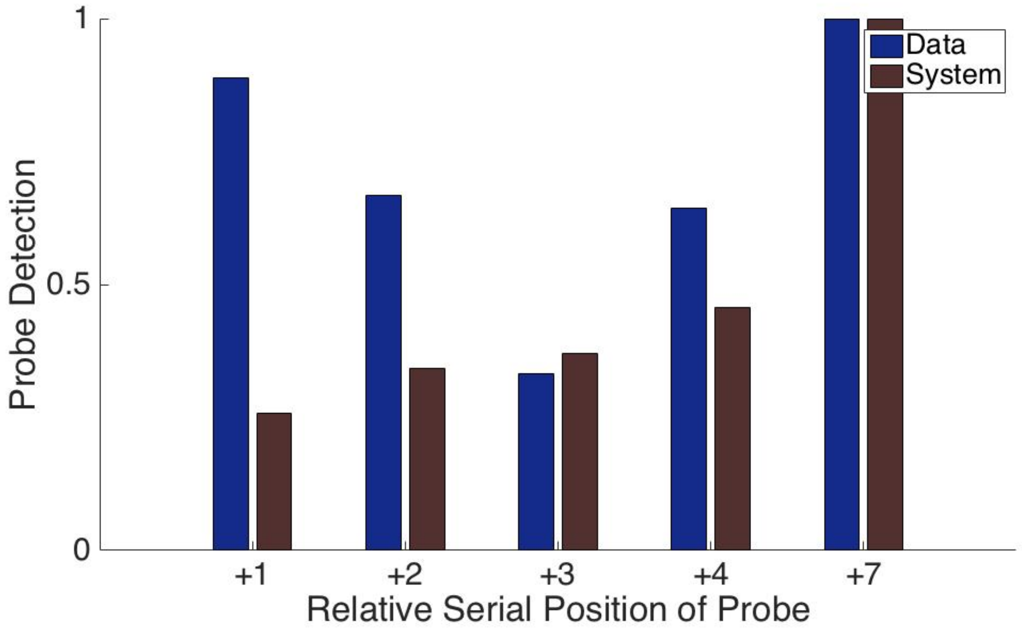


Figure 6.6: Comparison of the % Correct Probe Detection found in [57] reported as 'Data', with respect to the similarity reported as 'System.'

Summary This experiment is unique as it demonstrates a variety of CPs that can be "stitched" together to simulate larger and more complex experiments. Here, we have demonstrated that the Raymond et al. 1992 experiment can be composed of a detection followed by identification of the target followed by the detection of the probe. In addition, we have also highlighted how target identification processes can ballistically initiate an inhibitory memory consolidation signal as the source of interference the authors in the original experiment had found.

6.2 Neural Tuning Curve Experiments

In addition to testing the implemented CPs on visual attention experiments, we also wanted to test whether applying a subset of the CPs on the Visual Hierarchy result in neurally realistic tuning curves. Here we look at the effect of spatial and feature attention on the neural tuning curves and compare the findings to known neural tuning curves under these conditions. In order to record neural activation, we attach a probe in one sheet in layer V4 of the VH and plot one neurons activation across all conditions.

```
// Create a probe
Probe* probe = new Probe(tn3_network->getNetwork("VH"),
                         "V4_0_0");
// Attach the probe.
RIG()->addProbe(probe);
```

6.2.1 Spatial Attention

In a spatial attention paradigm, oriented stimuli are presented within the receptive field of the neuron being recorded in V4 area of macaque monkeys [40]. The monkeys are trained to either pay attention to the stimulus within or outside the receptive field in a match to sample task. The authors found that the effect of spatial attention was a multiplicative scaling of all orientations.

Figure 6.7 shows the sequence of images presented to the TarzaNN system reproduced from the original experiment by McAdams & Maunsell [40]. The fixation, delay and sample period were 500 ms each, while the test period could last up to 1000 ms. The experiment begins with the presentation of a fixation. The system is instructed to fixate here as only covert attention is allowed. Following this, a location cue indicated by a green circle is presented and the system uses this to build an attentional sample. During the inter-stimulus interval (ISI), a top down spatial bias is applied using the attentional sample as parameters. The target is then displayed and the system must now respond when it finds the target. This paradigm was run with orientation bars ranging from $[0° - 180°]$ in increments of $9°$ for a total of 20 orientation bars presented. In the attend toward and attend away conditions the target was presented at the cued and uncued location, respectively.

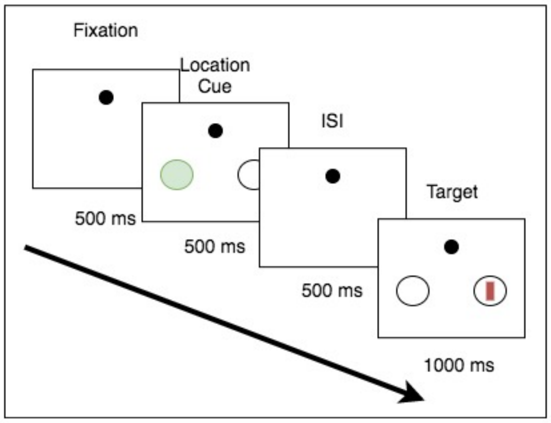


Figure 6.7: Spatial attention experimental sequence adapted from [40].

vTE CP Execution The vTE code listing for this experiment is shown in Listing 6.6. Note that the detection process here is used to ensure neural activations converge before terminating the experiment. In this simple experiment, the localization process is initiated to get the attentional sample from the VH. Unlike in the visual attention experiments, there is sufficient time for a full feedback traversal of the VH, thus a complete attentional sample is captured. The attentional sample is parameterized, then used to prime the VH.

```
void main() {
    // Variable Initialization
    bool detection;
    AttentionalSample* prime_as;
    // Display fixation image
    VAE()->waitForStimulus("Fixation");
    // Look at the centre fixation box.
    VAE()->FC()->setFHMBias({"location":"centre"},0.8);

    // A cue will appear for 500 ms then disappear.
    VAE()->waitForStimulus("Cue");

    prime_as = VAE()->VH()->getAS();
    // Transform the representation for spatial location only.
    transformASToLocationAS(prime_as);
    VAE->VH->Prime(prime_as);

    // A target will appear for 500 ms then disappear.
    VAE()->waitForStimulus("Target");

    // Detect the target
    detection = VAE()->VH()->Detection({"color":"red"})

    cout << "Target found: " << detection << "\n";
}
```

Listing 6.6: The commands executed by the vTE for simulating this experiment.

Results In Figure 6.8, we show that spatially priming the VH before target onset results in a multiplicative scaling effect as seen in the original experiment.

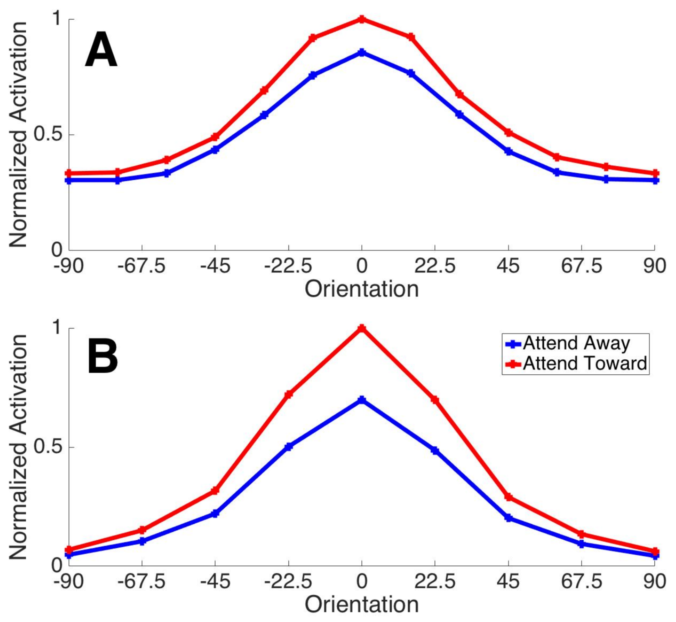


Figure 6.8: Spatial attention causes a multiplicative scaling of neural tuning curves. (A) The orientation tuning curves averaged across a population of V4 neurons reproduced from [40]. (B) TarzaNN 3.0 simulation results.

6.2.2 Feature Attention

Feature based attention is thought to occur from a different source than spatial based attention. Martinez-Treujilo & Treue [39] show the effects of feature based attention on neural tuning curves with directionally selective cells in Middle Temporal (MT) area of Macaque Monkeys. They note that the direction of motion of the attended stimuli resulted in different neural tuning curves of MT neurons with nearby features showing an enhancement while features farther away showed a suppressive effect.

McAdams & Maunsell [40] show the presence of orientation selective neurons in V4 and while in this experiment there was no sharpening of neural tuning curves found, this discrepancy may have been attributable to the differences in attentional strategy used [59]. In our experiment we qualitatively compare the results of a sharpening of the neural tuning curve of V4 neurons for feature based attention to those found in area MT in [39].

The experimental paradigm is almost identical to spatial attention except that here, the location cue is fixed across conditions and the neuron contralateral to the attended stimulus is recorded. In this way, the effects of spatial attention are controlled for. Feature based attention is always directed to the shown stimulus while the recorded neuron is sensitive only to a particular orientation. Attention was directed to either the fixation, or the stimulus in the contralateral receptive field. The key findings of the authors were that the multiplicative scaling due to feature based attention is proportional to the similarity of the attended feature with respect to the preferred feature of the neuron. This finding is also referred to as a sharpening of the neural tuning curve. The experimental paradigm is shown in Figure 6.9.

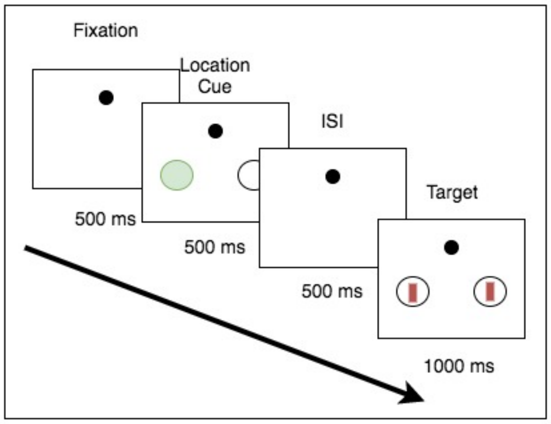


Figure 6.9: Feature attention experiment paradigm.

vTE CP Execution To run this experiment, we highlight some of the commands executed by the vTE in the code snippet shown in Listing 6.7. In the *attend same* condition, the system is manually primed to the orientation of the target (see Line 12 of code listing).

```
1   void main() {
2       // Variable Initialization
3       bool detection;
4       AttentionalSample* prime_as;
5       // Display fixation image
6       VAE()->waitForStimulus("Fixation");
7       // Look at the centre fixation box.
8       VAE()->FC()->setFHMBias({"location":"centre"},0.8);
9
10      // A cue will appear for 500 ms then disappear.
11      VAE()->waitForStimulus("Cue");
12
13      prime_as = VAE()->VH()->getAS();
14      // Transform the representation for spatial location only.
15      transformASToLocationAS(prime_as);
16      VAE()->VH()->Prime(prime_as,{"orientation":"X"});
17
18      // A target will appear for 500 ms then disappear.
19      VAE()->waitForStimulus("Target");
20
21      // Detect the target
22      detection = VAE()->VH()->Detection({"color":"red"})
23
24      cout << "Target found: " << detection << "\n";
}
```

Listing 6.7: The commands executed by the vTE for simulating this experiment.

Results Figure 6.10 compares the qualitative effect of feature based attention on neural response curves of one neuron in MT (Figure 6.10A reproduced from [39]) and V4 (Figure 6.10A). Here we show that the system is able to replicate the non-multiplicative scaling of the neural tuning curve. The data was generated by taking the neural activation at one time point after target detection when the neural activation's had converged.

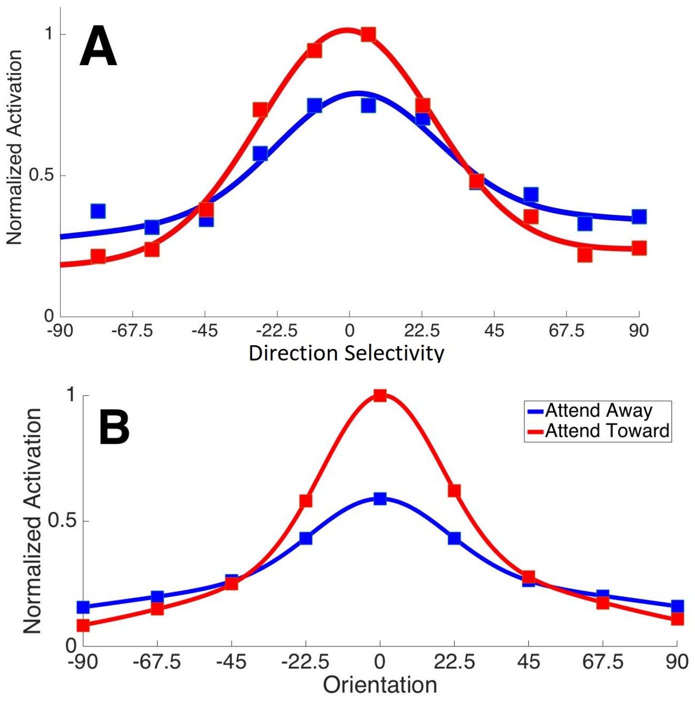


Figure 6.10: Feature based attention effects neural tuning curves in area V4. (A) The neural response curves recorded in layer MT of macaque monkeys from [39] and (B) from the system when attention was directed to the stimuli contralateral to the receptive field (Attend Toward) or toward the fixation (Attend Away).

7 Discussion and Conclusion

In this thesis we have introduced a Cognitive Programs Memory framework that acts as a "database" for storing Cognitive Programs which can be parameterized, reused and adapted for a variety of different visual tasks. As part of this framework, four Neural Primitives that function on the basis of transforming one representation to another are proposed that are not only biologically plausible but provide useful low level computations that can be sequenced together to form CPs. To allow CPs to have their full functionality, three other constituents were introduced: productions, control signals and parameterizations allowing us to create algorithms involving: top down control signals, selecting region of interest, dealing with covert and overt fixations, information routing, access to memory, matching to task and priming.

The motivation for introducing the CPM was two fold. First, there was no standardized way of executing a visual task in STAR, and designing algorithms on a task by task basis was difficult and slowed research and future development of ST. Second, taking inspiration from the neurophysiological literature, we wanted to design these algorithms around the idea of a basis set of elemental operations that can be combined together in different ways to provide a breadth of functionality. For this, we introduced the NPs.

To assess the CPM, we tested on three psychophysical experiments employing distinct processing strategies to demonstrate a variety of different CP functionalities. In addition, we also tested the CPs on a spatial and feature attention paradigm and showed that the system is able to replicate qualitatively similar neural tuning curves to neurophysiological data.

Despite this, there are several limitations to the CPM framework that can be expanded upon in future work. First, this framework relies on hard coded weights between any two Nodes that are connected by some NP operation. Although this simplifies the implementation, it significantly limits the functions that we can compute. As a solution, one may look at approximating these weights based on a desired function, much like how it is described in the NENGO framework [6]. Second, the execution of a CP requires a tremendous amount of parameterizations since each experiment contains stimuli at different scales and intensities requiring one to hand pick values such as θ for the $\theta - WTA$ algorithm. Furthermore, as part of the constituents of CPs we defined productions and manipulation of control signals using simple IF-THEN rules. Future work may look at encoding productions using neural representations and some form of a competitive WTA algorithm as theoretically defined in the human Turing machine [97] or as implemented in the NENGO framework [6]. Then, the manipulation of these control signals may be interpreted as a "firing" of some of these productions using a neural production system.

References

[1] Larry F Abbott. “Lapicque’s introduction of the integrate-and-fire model neuron (1907)”. In: *Brain research bulletin* 50.5 (1999), pp. 303–304.

[2] Fabrice Arcizet, Koorosh Mirpour, and James W Bisley. “A pure salience response in posterior parietal cortex”. In: *Cerebral Cortex* 21.11 (2011), pp. 2498–2506.

[3] Farhan Baluch and Laurent Itti. “Mechanisms of top-down attention”. In: *Trends in neurosciences* 34.4 (2011), pp. 210–224.

[4] Chiara Baston and Mauro Ursino. “A biologically inspired computational model of basal ganglia in action selection”. In: *Computational intelligence and neuroscience* 2015 (2015), p. 93.

[5] Andre M Bastos et al. “Canonical microcircuits for predictive coding”. In: *Neuron* 76.4 (2012), pp. 695–711.

[6] Trevor Bekolay et al. “Nengo: a Python tool for building large-scale functional brain models”. In: *Frontiers in neuroinformatics* 7 (2014), p. 48.

[7] Bruce Bobier, Terrence C Stewart, and Chris Eliasmith. “A unifying mechanistic model of selective attention in spiking neurons”. In: *PLoS computational biology* 10.6 (2014), e1003577.

[8] Rafal Bogacz. “Optimal decision-making theories: linking neurobiology with behaviour”. In: *Trends in cognitive sciences* 11.3 (2007), pp. 118–125.

[9] RhEInIschE FRIEDRIch-WILhELMs-UnIvERsIT^{2}AT Bonn. “VOCUS: A Visual Attention System for Object Detection and Goal-directed Search”. In: (2006).

[10] Conrado A Bosman and Francisco Aboitiz. “Functional constraints in the evolution of brain circuits”. In: *Frontiers in neuroscience* 9 (2015).

[11] Randy L Buckner, Jessica R Andrews-Hanna, and Daniel L Schacter. “The brain’s default network”. In: *Annals of the New York Academy of Sciences* 1124.1 (2008), pp. 1–38.

[12] Timothy J Buschman and Sabine Kastner. “From behavior to neural dynamics: an integrated theory of attention”. In: *Neuron* 88.1 (2015), pp. 127–144.

[13] Timothy J Buschman and Earl K Miller. “Top-down versus bottom-up control of attention in the prefrontal and posterior parietal cortices”. In: *science* 315.5820 (2007), pp. 1860–1862.

[14] Daniel P Buxhoeveden and Manuel F Casanova. “The minicolumn hypothesis in neuroscience”. In: *Brain* 125.5 (2002), pp. 935–951.

[15] Marisa Carrasco. “Visual attention: The past 25 years”. In: *Vision research* 51.13 (2011), pp. 1484–1525.

[16] Patrick Cavanagh. “Attention routines and the architecture of selection”. In: (2004).

[17] Marlene R Cohen and John HR Maunsell. “Attention improves performance primarily by reducing interneuronal correlations”. In: *Nature neuroscience* 12.12 (2009), pp. 1594–1600.

[18] Stanislas Dehaene et al. “The neural representation of sequences: from transition probabilities to algebraic patterns and linguistic trees”. In: *Neuron* 88.1 (2015), pp. 2–19.

[19] Robert Egly, Jon Driver, and Robert D Rafal. “Shifting visual attention between objects and locations: evidence from normal and parietal lesion subjects.” In: *Journal of Experimental Psychology: General* 123.2 (1994), p. 161.

[20] Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems.* MIT press, 2004.

[21] Charles L Folk, Roger W Remington, James C Johnston, et al. “Involuntary covert orienting is contingent on attentional control settings”. In: *Journal of Experimental Psychology Human Perception and Performance* 18 (1992), pp. 1030–1030.

[22] Michael J Frank and David Badre. “Mechanisms of hierarchical reinforcement learning in corticostriatal circuits 1: computational analysis”. In: *Cerebral cortex* 22.3 (2011), pp. 509–526.

[23] David J Freedman et al. “Visual categorization and the primate prefrontal cortex: neurophysiology and behavior”. In: *Journal of Neurophysiology* 88.2 (2002), pp. 929–941.

[24] Pascal Fries et al. “Modulation of oscillatory neuronal synchronization by selective visual attention”. In: *Science* 291.5508 (2001), pp. 1560–1563.

[25] Karl J Friston. “Functional and effective connectivity: a review”. In: *Brain connectivity* 1.1 (2011), pp. 13–36.

[26] CG Gross et al. “Cortical visual areas of the temporal lobe”. In: *Multiple visual areas.* Springer, 1981, pp. 187–216.

[27] David H Hubel and Torsten N Wiesel. “Ferrier lecture: Functional architecture of macaque monkey visual cortex”. In: *Proceedings of the Royal Society of London B: Biological Sciences* 198.1130 (1977), pp. 1–59.

[28] Stanley Jacobson and Elliott M Marcus. *Neuroanatomy for the Neuroscientist.* Springer Science & Business Media, 2011.

[29] Raymond M Klein. "Inhibition of return". In: *Trends in cognitive sciences* 4.4 (2000), pp. 138–147.

[30] Iuliia Kotseruba. "Visual Attention in Dynamic Environments and Its Application To Playing Online Games". In: (2016).

[31] Wouter Kruijne and John K Tsotsos. *Visuo-cognitive routines: reinterpreting the theory of visual routines as a framework for visual cognition.* Tech. rep. TR CSE-2011-05. Toronto, ON: Department of Computer Science and Eng., York University, 2011.

[32] Toni Kunic. "Cognitive Program Compiler". In: (2017).

[33] José L Lanciego, Natasha Luquin, and José A Obeso. "Functional neuroanatomy of the basal ganglia". In: *Cold Spring Harbor perspectives in medicine* 2.12 (2012), a009621.

[34] Matthew Larkum. "A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex". In: *Trends in neurosciences* 36.3 (2013), pp. 141–151.

[35] Kyoung-Min Lee, Kyung-Ha Ahn, and Edward L Keller. "Saccade generation by the frontal eye fields in rhesus monkeys is separable from visual detection and bottom-up attention shift". In: *PloS one* 7.6 (2012), e39886.

[36] Alianna J Maren, Craig T Harston, and Robert M Pap. *Handbook of neural computing applications.* Academic Press, 2014.

[37] Henry Markram. "A network of tufted layer 5 pyramidal neurons." In: *Cerebral cortex (New York, NY: 1991)* 7.6 (1997), pp. 523–533.

[38] David Marr and Tomaso Poggio. "A computational theory of human stereo vision". In: *Proceedings of the Royal Society of London B: Biological Sciences* 204.1156 (1979), pp. 301–328.

[39] Julio C Martinez-Trujillo and Stefan Treue. "Feature-based attention increases the selectivity of population responses in primate visual cortex". In: *Current Biology* 14.9 (2004), pp. 744–751.

[40] Carrie J McAdams and John HR Maunsell. "Effects of attention on orientation-tuning functions of single neurons in macaque cortical area V4". In: *Journal of Neuroscience* 19.1 (1999), pp. 431–441.

[41] AR McIntosh and F Gonzalez-Lima. "Network interactions among limbic cortices, basal forebrain, and cerebellum differentiate a tone conditioned as a Pavlovian excitor or inhibitor: fluorodeoxyglucose mapping and covariance structural modeling". In: *Journal of Neurophysiology* 72.4 (1994), pp. 1717–1733.

[42] Maria Medalla and Helen Barbas. “Synapses with inhibitory neurons differentiate anterior cingulate from dorsolateral prefrontal pathways associated with cognitive control”. In: *Neuron* 61.4 (2009), pp. 609–620.

[43] Ashesh D Mehta, Istvan Ulbert, and Charles E Schroeder. “Intermodal selective attention in monkeys. I: distribution and timing of effects across visual areas”. In: *Cerebral Cortex* 10.4 (2000), pp. 343–358.

[44] Earl K Miller and Timothy J Buschman. “Cortical circuits for the control of attention”. In: *Current opinion in neurobiology* 23.2 (2013), pp. 216–222.

[45] Earl K Miller, Cynthia A Erickson, and Robert Desimone. “Neural mechanisms of visual working memory in prefrontal cortex of the macaque”. In: *Journal of neuroscience* 16.16 (1996), pp. 5154–5167.

[46] Tirin Moore and Katherine M Armstrong. “Selective gating of visual signals by microstimulation of frontal cortex”. In: *Nature* 421.6921 (2003), pp. 370–373.

[47] Sancho I Moro et al. “Neuronal activity in the visual cortex reveals the temporal order of cognitive operations”. In: *Journal of Neuroscience* 30.48 (2010), pp. 16293–16303.

[48] Hermann J Müller and Patrick M Rabbitt. “Reflexive and voluntary orienting of visual attention: time course of activation and resistance to interruption.” In: *Journal of Experimental psychology: Human perception and performance* 15.2 (1989), p. 315.

[49] Alexander Naka and Hillel Adesnik. “Inhibitory circuits in cortical layer 5”. In: *Frontiers in neural circuits* 10 (2016).

[50] Walle JH Nauta, Michael Feirtag, and Carol Donner. *Fundamental neuroanatomy.* Freeman New York, 1986.

[51] Joel Norman. “Two visual systems and two theories of perception: An attempt to reconcile the constructivist and ecological approaches”. In: *Behavioral and brain sciences* 25.1 (2002), pp. 73–96.

[52] Ioan Opris, Andrei Barborica, and Vincent P Ferrera. “Microstimulation of the dorsolateral prefrontal cortex biases saccade target selection”. In: *Journal of Cognitive Neuroscience* 17.6 (2005), pp. 893–904.

[53] Randall C O’Reilly. “Biologically based computational models of high-level cognition”. In: *science* 314.5796 (2006), pp. 91–94.

[54] Randall C O'Reilly and Michael J Frank. "Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia". In: *Neural computation* 18.2 (2006), pp. 283–328.

[55] Hae-Jeong Park and Karl Friston. "Structural and functional brain networks: from connections to cognition". In: *Science* 342.6158 (2013), p. 1238411.

[56] Michael I Posner and Yoav Cohen. "Components of visual orienting". In: *Attention and performance X: Control of language processes* 32 (1984), pp. 531–556.

[57] Jane E Raymond, Kimron L Shapiro, and Karen M Arnell. "Temporary suppression of visual processing in an RSVP task: An attentional blink?" In: *Journal of experimental psychology: Human perception and performance* 18.3 (1992), p. 849.

[58] John H Reynolds, Leonardo Chelazzi, and Robert Desimone. "Competitive mechanisms subserve attention in macaque areas V2 and V4". In: *Journal of Neuroscience* 19.5 (1999), pp. 1736–1753.

[59] John H Reynolds and David J Heeger. "The normalization model of attention". In: *Neuron* 61.2 (2009), pp. 168–185.

[60] Pieter R Roelfsema. "Elemental operations in vision". In: *Trends in cognitive sciences* 9.5 (2005), pp. 226–233.

[61] Pieter R Roelfsema, Paul S Khayat, and Henk Spekreijse. "Subtask sequencing in the primary visual cortex". In: *Proceedings of the National Academy of Sciences* 100.9 (2003), pp. 5467–5472.

[62] Andrew F Rossi et al. "Top–down attentional deficits in macaques with lesions of lateral prefrontal cortex". In: *Journal of Neuroscience* 27.42 (2007), pp. 11306–11314.

[63] Albert L Rothenstein and John K Tsotsos. "Attentional modulation and selection–an integrated approach". In: *PloS one* 9.6 (2014), e99681.

[64] Albert L Rothenstein, Andrei Zaharescu, and John K Tsotsos. "TarzaNN: A general purpose neural network simulator for visual attention modeling". In: *WAPCV*. Springer. 2004, pp. 159–167.

[65] Yuri B Saalmann and Sabine Kastner. "Cognitive and perceptual functions of the visual thalamus". In: *Neuron* 71.2 (2011), pp. 209–223.

[66] Yuri B Saalmann et al. "The pulvinar regulates information transmission between cortical areas based on attention demands". In: *Science* 337.6095 (2012), pp. 753–756.

[67] Emilio Salinas and Terrence J Sejnowski. “Impact of correlated synaptic input on output firing rate and variability in simple neuronal models”. In: *Journal of neuroscience* 20.16 (2000), pp. 6193–6209.

[68] Jeffrey D Schall. “Neural basis of deciding, choosing and acting”. In: *Nature Reviews Neuroscience* 2.1 (2001), pp. 33–42.

[69] Nicolas W Schuck et al. “Human orbitofrontal cortex represents a cognitive map of state space”. In: *Neuron* 91.6 (2016), pp. 1402–1412.

[70] Rakesh Sengupta et al. “Attentional blink as a product of attentional control signals: A computational investigation”. In: *Journal of Vision* 17.10 (2017), pp. 1197–1197.

[71] Zhengwei Shao and Andreas Burkhalter. “Role of GABA B receptor-mediated inhibition in reciprocal interareal pathways of rat visual cortex”. In: *Journal of neurophysiology* 81.3 (1999), pp. 1014–1024.

[72] Shazia Veqar Siddiqui et al. “Neuropsychology of prefrontal cortex”. In: *Indian journal of psychiatry* 50.3 (2008), p. 202.

[73] R Angus Silver. “Neuronal arithmetic”. In: *Nature reviews. Neuroscience* 11.7 (2010), p. 474.

[74] Olaf Sporns and Rolf Kötter. “Motifs in brain networks”. In: *PLoS biology* 2.11 (2004), e369.

[75] Olaf Sporns, Giulio Tononi, and Rolf Kötter. “The human connectome: a structural description of the human brain”. In: *PLoS computational biology* 1.4 (2005), e42.

[76] “The American Heritage® Science Dictionary”. In: (Dec. 2017). URL: `http://www.dictionary.com/browse/atomic`.

[77] PHE Tiesinga et al. “Information transfer in entrained cortical neurons”. In: *Network: Computation in Neural Systems* 13.1 (2002), pp. 41–66.

[78] Hyoe Tomita et al. “Top-down signal from prefrontal cortex in executive control of memory retrieval”. In: *Nature* 401.6754 (1999), pp. 699–703.

[79] Giulio Tononi, Olaf Sporns, and Gerald M Edelman. “A measure for brain complexity: relating functional segregation and integration in the nervous system”. In: *Proceedings of the National Academy of Sciences* 91.11 (1994), pp. 5033–5037.

[80] Anne M Treisman and Garry Gelade. “A feature-integration theory of attention”. In: *Cognitive psychology* 12.1 (1980), pp. 97–136.

[81] Stefan Treue and Julio C Martinez Trujillo. “Feature-based attention influences motion processing gain in macaque visual cortex”. In: *Nature* 399.6736 (1999), pp. 575–579.

[82] John K Tsotsos. *A computational perspective on visual attention.* MIT Press, 2011.

[83] John K Tsotsos. “Complexity Level Analysis Revisited: What Can 30 Years of Hindsight Tell Us about How the Brain Might Represent Visual Information?” In: *Frontiers in psychology* 8 (2017), p. 1216.

[84] John K Tsotsos and Wouter Kruijne. “Cognitive programs: software for attention’s executive”. In: *Frontiers in psychology* 5 (2014).

[85] John K Tsotsos et al. “The different stages of visual recognition need different attentional binding strategies”. In: *Brain research* 1225 (2008), pp. 119–132.

[86] John Tsotsos, Iuliia Kotseruba, and Calden Wloka. “A Focus on Selection for Fixation”. In: *Journal of Eye Movement Research* 9.5 (2016).

[87] Federico E Turkheimer et al. “The brain’s code and its canonical computational motifs. From sensory cortex to the default mode network: A multi-scale model of brain function in health and disease”. In: *Neuroscience & Biobehavioral Reviews* 55 (2015), pp. 211–222.

[88] Shimon Ullman. “Visual routines”. In: *Cognition* 18.1 (1984), pp. 97–159.

[89] Nobuhiko Wagatsuma et al. “Spatial and feature-based attention in a layered cortical microcircuit model”. In: *PloS one* 8.12 (2013), e80788.

[90] Claire Wardak et al. “Contribution of the monkey frontal eye field to covert visual attention”. In: *Journal of Neuroscience* 26.16 (2006), pp. 4228–4235.

[91] Benjamin Wilson, William D Marslen-Wilson, and Christopher I Petkov. “Conserved sequence processing in primate frontal cortex”. In: *Trends in neurosciences* (2017).

[92] Thilo Womelsdorf and Stefan Everling. “Long-range attention networks: circuit motifs underlying endogenously controlled stimulus selection”. In: *Trends in neurosciences* 38.11 (2015), pp. 682–700.

[93] Thilo Womelsdorf et al. “Dynamic circuit motifs underlying rhythmic gain control, gating and integration”. In: *Nature neuroscience* 17.8 (2014), pp. 1031–1039.

[94] Takao Yamasaki et al. “Motion perception in autism spectrum disorder”. In: *Advances in psychology research* 82 (2011), pp. 197–211.

[95] Weilie Yi and Dana Ballard. “Recognizing behavior in hand-eye coordination patterns”. In: *International Journal of Humanoid Robotics* 6.03 (2009), pp. 337–359.

[96] Ariel Zylberberg et al. “A neuronal device for the control of multi-step computations”. In: *Papers in physics* 5.2 (2013), pp. 1–15.

[97] Ariel Zylberberg et al. “The human Turing machine: a neural framework for mental programs”. In: *Trends in cognitive sciences* 15.7 (2011), pp. 293–300.

Appendix A Implementation

A.1 TN3Network

Three kinds of Nodes may be created with a specification of its name and size (lines 2-4) while a Neural Primitive is instantiated as shown on line 6 in Listing A.1.

```
1  // Creating the three kinds of Nodes with a size of 5x5.
2  DEFAULT_NODE* node = new DEFAULT_NODE(string name,int w, int h);
3  CONTEXT_NODE* node = new CONTEXT_NODE(string name,int w, int h);
4  MEMORY_NODE* node = new MEMORY_NODE(string name,int w, int h);
5  // Abstract Class Constructor
6  NeuralPrimitive* op = new NeuralPrimitive(FcnGen* x, FcnGen* y,
       ConnectivityType type);
```

Listing A.1: Node Instantiation with a specified width (w) and height (h).

Listing A.2 shows the TN3Network for priming the VH with an AS. Line 2 is used to create a new instance of the priming network, while line 3 is used to retrieve all of the Nodes in the VH. Creation of the Nodes and NP operations then occurs by first retrieving the *width* and *height* of the Nodes (lines 7 and 8) followed by dynamic specification of the Node name (lines 9 and 10). After Node creation (line 11), the NP operation is created (line 14) with a parameter specification provided by the *FcnGen* class (line 13). The input and VH node as well as the NP operation are then added to the network (lines 15-17). The initialized TN3Network is then returned (line 20).

```
1  TN3Network* initPrimeAS(TN3Network* vh_network){
2      TN3Network* network = new TN3Network("PrimeAS");
3      vector<NODE*> vh_nodes = vh_network->getNodes(vh_network);
4      // Create a NODE for each VH sheet.
5      for (int i = 0; i <vh_nodes.size();i++){
6          NODE* vh_node = vh_nodes.at(i);
7          int width = vh_node->getOutput()->getXSize();
8          int height = vh_node->getOutput()->getYSize();
9          string node_name = network_name + "_" + vh_node->getName();
10         string input_node_name = node_name + "input_node";
11         CONTEXT_NODE* input_node = new CONTEXT_NODE(input_node_name,
12                                                      width,height);
13         FcnGen* weight_fcn = new FcnGen();
14         fcn->setLinear(1.0);
15         BiasControl* bc = new BiasControl(fcn,fcn,ONE_TO_ONE);
16         network->addNode(input_node);
17         network->addNode(VH_node);
18         network->addOperation(input_node,VH_node,bc);
19     }
20     // Return the network.
21     return network;
22 }
```

Listing A.2: Network creation for priming with an AS algorithm.

Networks can then be added iteratively to other networks by either retrieving them from storage in the **TN3StorageNetwork** class or from manually defined TN3Networks (lines 3-8). The network is then initialized (line 10) as shown in Listing A.3.

```
 1 // Global network to encapsulte subnetworks
 2 TN3Network* global_network = new TN3Network("GLOBAL");
 3 // Initialize VH with default parameters
 4 TN3Network* vh_network = TN3StorageNetwork::getVHNetwork();
 5 // Initialize Priming Network
 6 TN3Network* prime_as_network = TN3StorageNetwork::PrimeAS(vh_network)
       ;
 7 // Add the networks to the global network
 8 global_network->addNetwork(as_prime_network);
 9 global_network->addNetwork(vh_network);
10 // Initialize the Network
11 global_network->init()
```

Listing A.3: A toy network showing how subnetworks are added to the main network and passed to the runner for updating.

In Section 4, we described several Cognitive Programs, all of which rely on a set of fixed networks to be initialized before a visual task can be executed. The functions below in the **TN3NetworkStorage** class show the prebuilt networks used for this thesis. Each of which returns a **TN3Network*** as shown in Table A.2. Notice that the **TN3Neworks** rely on other networks to be initialized before their own initialization. This is also due to the connectivity defined in the complete STAR architecture (see Figure C.1.

Base	**Arguments**	**Description**
TN3NetworkStorage:: getVHNetwork	N/A	Initializes the VH network using default parameters. See Appendix C.
TN3NetworkStorage:: getVWM	TN3Network* *vh_network*	Requires an existing instance of a VH network to initialize.
TN3NetworkStorage:: getPrimeAS	TN3Network* *vh_network*	See Above.
TN3NetworkStorage:: getPrimeLocAndFeature	TN3Network* *vh_network*, float *min_bias_weight*	A minimum bias weight specifies the maximum suppression allowed.
TN3NetworkStorage:: getTWM	TN3Network* *vh_network*, int *num_slots*	The number of slots indicates the maximum number of attentional samples that we can store at any given time.

TN3NetworkStorage:: getMatchDetection	TN3Network* *vh_ network*, TN3Network* *vwm_ network*, TN3Network* *twm_ network*	Requires the existence of the VH, vWM and tWM networks.
TN3NetworkStorage:: getConsolidationNetwork	TN3Network* *vh_ network*	Requires the existence of the VH network.

Table A.2: TarzaNN 3.0 pre-stored networks for initialization. See Section 4 for more details.

A.2 CPM-ME Base Methods

Base Method	**Implementation**	**Description**
Gate <source node> to <destination node>	enableGating(NeuralPrimitive* *np*, bool *value*)	Enables or disables all the nodes in the specified subgraph network.
Read <node>	readNode(Node* *node*)	Reads the current Node state.
Write <node>	writeNode(Node* *node*, Matrix* *write_ data*)	Uses the **FcnGen** class to set the input to a context node.

Table A.3: "Access to representation" Base Method implementation.

Base Method	**Implementation**	**Description**
Set <Decision Criteria> Params	**See Table A.10.**	
Set <Decision Criteria> Params		
Set <Decision Criteria> Params		
Set <Mem Consolidation> Params		
Load <AS> to <tWM,vWM>	loadASToTWM(AttentionalSample* *as*, TN3Network* *dst_ net*)	Loads the attentional sample. We may store multiple AS's in tWM, each accessible by its index.

Table A.4: "Parameter Specification" Base Method implementation.

Base Method	Implementation
Feedforward (FF) retinal Signal	runner->update(int *time*)
Select cFOA	vae->getIsWinnerSelected(int *Layer*)
Apply suppressive surround (SS)	Internally computed and beyond the scope of this thesis.
Disengage attention	See Table A.10.
Do Location Inhibition of Return (L-IOR)	vae->doLIOR();
Reset <vWM,tWM>	vae->resetVWM() or vae->resetTWM()
Select <Layer Name>	BaseMethod::selectLayer (TN3Network* *network*, string *name*)
Select <NEXT Layer>	BaseMethod::selectNextLayer (TN3Network* *network*, string *name*)

Table A.5: “Executing a Process” Base Method implementation.

Base Method	Implementation
Is the winner found?	vae->isWTAFound()
Are there more layers?	vae->isNextLayer()
Is there AS for this layer?	isASExist(AttentionalSample* *as*)

Table A.6: “Queries” Base Method implementation.

A.3 Recognition Methods

Base	Arguments	Description
void VH()->Prime	vector<pair<string, string» *params*	Returns when priming complete down all layers of the VH or if the process is terminated earlier.
void VH()->Prime	AttentionalSample* *as*	See above.
void VH()->Prime	vector<pair<string, string» *params*, AttentionalSample* *as*	In the case when we want to prime with an Attentional Sample and something else (such as color). Useful for the Folk et al experiment (see Section 6.1.2) where we have to prime for the location and the target stimulus color.
boolVH()->Detection	int TYPE = {FEATURE_DETECTION, OBJECT_DETECTION, LOCATION_DETECTION}	Detection using an attentional sample (AS) with a specification of the kind of detection. Assumes the tWM contains the AS.
intVH()->Recognition	int TYPE	See above.
int VH()->Categorization	int TYPE	See above.
int VH()->Identification	int TYPE	See above.
bool VH()->Detection	map<string, string> *class_params*	Specification of feature or location we wish to detect.

int VH()->Recognition	map<string, string> *class_params_a*, map<string, string> *class_params_b*	Two classes, neither of which are noise. Returns a int representing class selection. 0 = class A, 1 = class B
int VH()->Categorization	vector<map<string, string»	N classes, neither of which are noise.
int VH()->Identification	vector<map<string, string»	See above.
AttentionalSample* VH()->Localization	N/A	Gets the FOA from each layer of the VH and returns it as an attentional sample.
AttentionalSample* VH()->getAS	N/A	Identical to VH.Localization
void FC()->setFHMBias	AttentionalSample* *as*, float *td_bias*	Bias's the FHM using an AS as input. Top down bias value is used to weigh top down influences with respect to bottom up influences. See Section 4.3.8 for more information.

Table A.8: Complete list of Cognitive Program methods, all of which are accessible through the VAE().

```
// Feature Priming
VH()->prime({"orientation":"0"})
// Detection for two features: Orientation and color
VH()->Detection({"orientation":"0","color":"red"})
// Detection using an AS
VH()->Detection(OBJECT_DETECTION)
// Recognition of one of two classes. Neither of which is noise
VH()->Recognition({"orientation":"0","color":"red"}, {"orientation":"
    0","color":"green"} )
// Returning an Attentional Sample from layer 'V1'
VH()->getLayer("V1")
// Setting a FHM for the center
FC()->setFHMBias({"location": "center"})
```

Listing A.4: Example CPM method calls with task specification

A.4 Auxiliary Functions

Base	**Arguments**	**Description**
VAE()->VH()->setDetectionCriteria	float *lowThresh*, float *upperThresh*, int *minTime*, int *maxTime*	Specifies the minimum and maximum thresholds for detection and the time range the detection must be made. See equation 4.5. If a minimum and maximum time has not been specified, the detection criteria only relies on the threshold.
VAE()->VH()-> setRecognitionCriteria	vector< struct *decision_criteria*>	Takes in a vector of *struct* for the decision criteria. Vector must only contain two elements given the definition of a Recognition task. See Section 4.3.6 for more information.
VAE()->VH()-> setIdentificationCriteria	See above.	Takes in a vector of *struct* for the decision criteria.
VAE()->VH()-> setMemConsolidation Params	int *max_inhibition*, float *duration*	Specifies the initial maximum inhibition within a range of [0.0, 1.0] to apply for a predetermined duration. See Section 4.3.6 for more information.
transformAS toLocationAS	AttentionalSample* *as*	Transforms an attentional sample to represent only location information. See Section 4.3.5.
VAE()-> waitForStimulus	string *stimulus_name*	Waits until the stimulus with the provided name appears before continuing CP execution.
VAE()-> disengageAllAttentionMechanisms	None	Disengages attention mechanisms.
RIG()-> addStimulus	string *file_name*, string *stimulus_name*, int *onset_duration*	Adds the stimulus with the given file name to the list of stimuli to present to the system for a given duration.

Table A.10: Auxillary TN3Runner functions

A.5 Function Generator

A function generator is used to test the system based on some hypothetical input and is implemented using the *FcnGen* class. Furthermore, it also serves to define the weight matrix ϕ shown in equation A.1 connecting two representations together. Note that in C.N.N. and deep learning approaches the weight function is learnt through back propagation [36], while in the Neural Engineering Framework [20] the weight function is approximated based on an arbitrary function they wish to approximate. Both of these aspects is beyond the scope of this thesis work. Instead, we allow this to be a user set parameter and leave it open for future work for automatically determining appropriate values.

$$\phi = \begin{bmatrix} \gamma_{1,1,k_{1,1}} g_{1,1,k_{1,1}} & \cdots & \gamma_{1,B,k_{1,B}} g_{1,B,k_{1,B}} \\ \vdots & \ddots & \ldots \\ \gamma_{A,1,k_{A,1}} g_{A,1,k_{A,1}} & \cdots & \gamma_{A,B,k_{A,B}} g_{A,B,k_{A,B}} \end{bmatrix} \tag{A.1}$$

This class provides a useful set of predefined functions; point wise, linear, gaussian, gabor, sin, cos and tan to simulate a variety of inputs. Rather than having the output of some other node as its input, a function generator is used to generate some hypothetical function. For example, top down task dependent bias's may be converted into inputs to another node to simulate top down control. Here we do not argue that this is biologically plausible, instead it is provided as a means of modeling a neurons behavior as a function of its input. NENGO [6] a popular neural simulator based on the NEF introduces a similar concept. Since this is implemented in TarzaNN, a neural simulator would not be complete without a way of simulating input. It consists of the following functions:

1. **void setLinear (float value):** Sets the function to a constant value.
2. **void setExpDecay (int begin_time, int INHIBITION_LENGTH):** Sets the function to an exponential decay that begins at a specified time with respect to the Neural Primitives clock time.
3. **void setPoint (int k):** Set's a point where the value is 1. All other points have a value of 0.
4. **void setGauss (float sigma, float center)**
5. **void setGabor (float lambda, float orientation, float phase, float sigma, float aspect_ratio)**
6. **void setSin/setCos/setTan (float amplitude, float phase)**
7. **float compute (int k):** Computes the function at k. k may represent the spatial position or time.

float compute (int k) is the key function in this class responsible for computing the specified function. In later sections, we will see how this is used by our Nodes and NP operations.

Appendix B Supplementary Results and Figures

This section includes supplementary experimental data, results and figures that were not included in the Methods and Evaluation Section.

B.1 Copying to vWM

Figure B.1 shows the neural activation of one neuron in each layer of the VH and the corresponding neuron in the vWM where the contents are copied to. The dotted blue vertical line indicates the time contents of VH were gated to vWM.

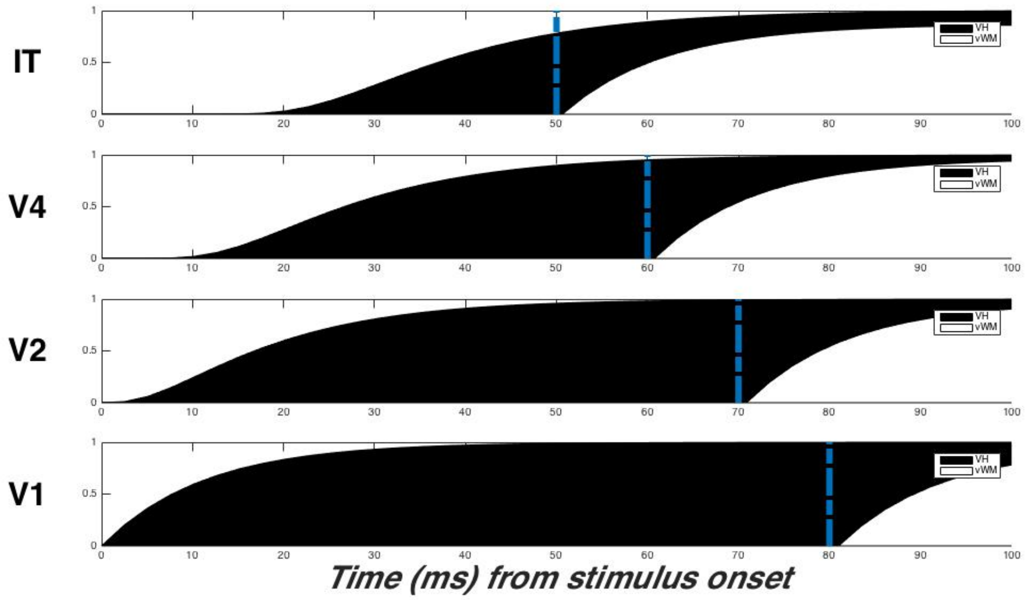


Figure B.1: Copying the contents of the VH to the vWM. Plot showing a chosen neurons neural activations at each layer of the VH. Dotted blue vertical line indicates the time the contents were gated to vWM.

B.2 Building an AS

Figure B.2 shows the neural activation map of three layers of the VH that have been gated to the vWM using the gating scheme shown in Figure B.1. Sheet 1 and Sheet 2 represent two feature maps within the same layer of the VH out of the total of 12 sheets per layer in our full network. Here, these two sheets are shown to indicate how the highest neural activation does not necessarily reflect the winner chosen at the top of the VH.

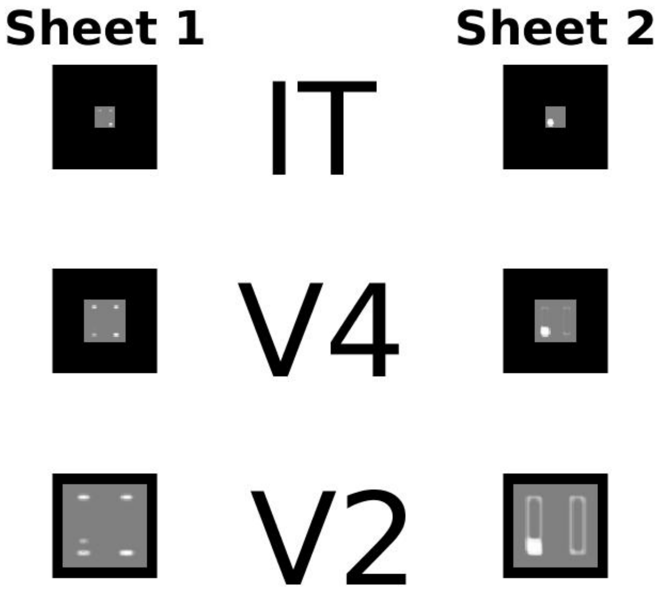


Figure B.2: Building an Attentional Sample by gating contents of VH to vWM. Two sheets are shown here for clarity and activation maps are displayed proportional to the size of the VH layer.

B.3 Transforming an AS

Figure B.3 shows how Figure B.2 can be transformed to represent spatial location only. Here, the winning region consisting of the set of neurons that contributed to the selection of the winner at the top of the VH are determined. This winning region is set to a maximum value since the signal is allowed to propagate unattenuated.

1. *Spatial AS*: If we wish to only prime for a certain location without worrying about features, then a gaussian around the winner region is selected as the attentional beam pass. This winner region is all set to a maximum bias value such that none of this location is inhibited.

2. *Feature AS*: If we wish to attend to a certain feature such as a color without the need for location information, we can parameterize our AS accordingly. This transformation sets the AS bias value parameterization to 1.0 for the sheets representing the feature with the maximum activation while all other sheets are set to a predefined value (parameterized during run time).(i.e. “the object is red”).

3. *Object AS*: The captured AS contains both location and feature information about the object. When both pieces of information are relevant, no further parameterization is necessary and the AS can be passed to the priming method directly.

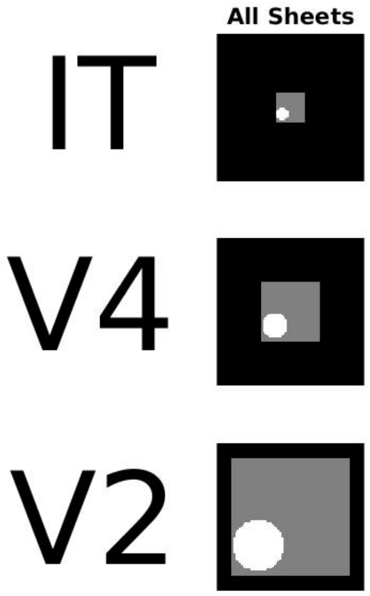


Figure B.3: Transforming the AS to represent spatial information only.

B.4 Priming the VH

Recall that priming the VH can occur via some top down user specified parameters or with the use of an attentional sample Figure B.4 shows the neural activation of one neuron in each layer of the VH. The dotted blue vertical line indicates the time the attentional sample was used to prime the VH.

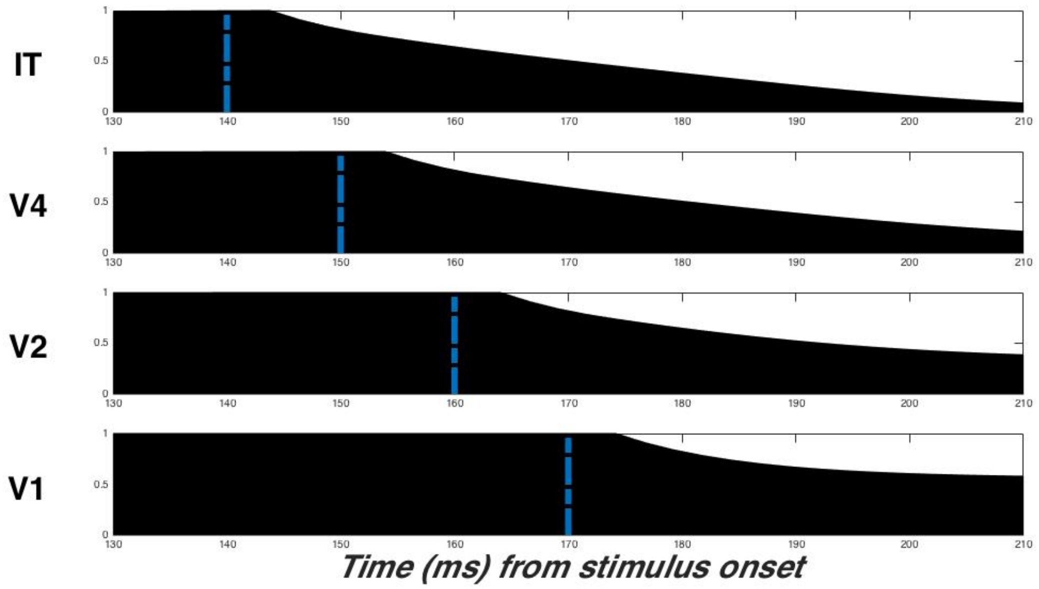


Figure B.4: The neural activation of one neuron in each layer of the VH during priming the VH using an attentional sample. The dotted vertical blue lines indicate the time the attentional sample was used to bias the VH at the given layer.

B.5 Memory Consolidation Signal

Figure B.5 shows the neural activation of one neuron as a function of the relative serial position of the probe with respect to the target. A probe presented after the target is likely to be suppressed due to a top down memory consolidation signal. Here, the memory consolidation signal is modeled as an exponential decay, shown as a dotted line in the figure. The dotted vertical lines represents the probe onset time. The distance between any two vertical lines is 90 ms as in the original experiment. The memory consolidation signal was parameterized to have a decay period of 500 ms.

When generating the figure, no other stimulus properties were changed other than the serial position of the probe. Similar results were found in [70].

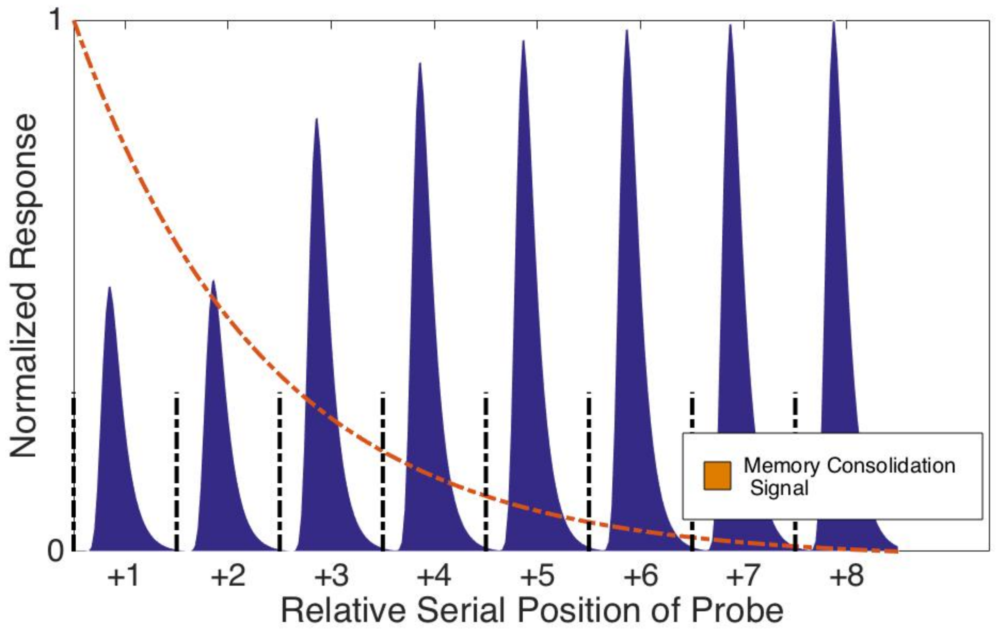


Figure B.5: Neural activation of one neuron in IT plotted in response to the probe. The dotted line shows the degree of the top down inhibitory signal applied

B.6 Egly & Driver 1994

The Egly & Driver 1994 experiment discussed in Section 6.1.1 is presented in detail here. For the following figures, only two of the twenty-four feature maps of each of the VH layers are shown for clarity. On the left and right hand side of the figures, a feature map selective for orientations of 0° and 90° respectively are shown.

The experiment begins with the presentation of a visual cue to the system (see Figure B.6). The system correctly identifies the cue as shown by the whiter areas indicating a higher neural activation than the surrounding regions.

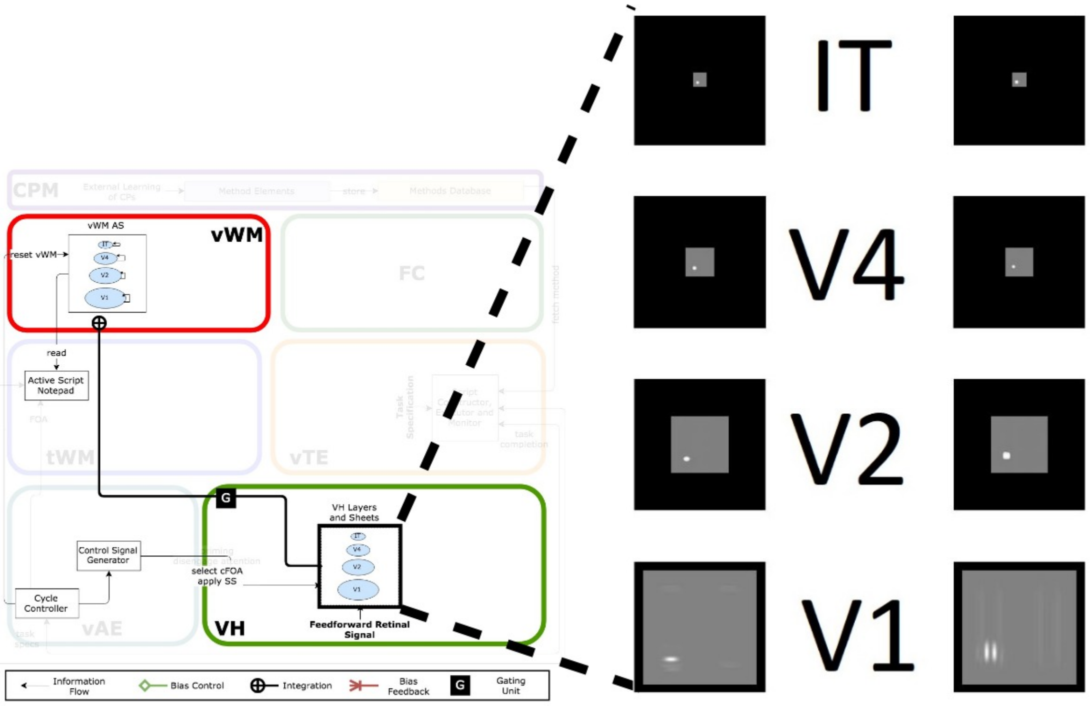


Figure B.6: Input cue presented and processed by the VH. Whiter areas indicate neurons with higher activations.

During the presentation of the cue, the CP for retrieving an AS from the VH is called. This initiates the process of storing information from the VH to the vWM using an Integration Neural Primitive operation (see Section 4.3.2).

```
prime_as = VAE()->VH()->getAS();
```

The AS is then transformed to represent location information only and abstracts away any feature information since we only want to use the AS to prime the system of the location it should expect the target will appear. The resulting AS is shown in Figure B.7.

```
transforAStoLocationAS(prime_as);
```

Figures B.8 and B.9 show the VH processing the target stimulus without and with priming respectively. Figure B.8 is shown here for comparison purposes only. Notice that areas away from the primed zone are inhibited in Figure B.9.

In the previous two Figures, we have only shown stimulus input for a correctly cued target. However, the system also processes input for an incorrectly cued target - when the filled square on one of the ends of the rectangle appears somewhere other than the cued region. Figure B.10 shows the contents of the vWM of two feature maps in the *IT* layer during the presentation

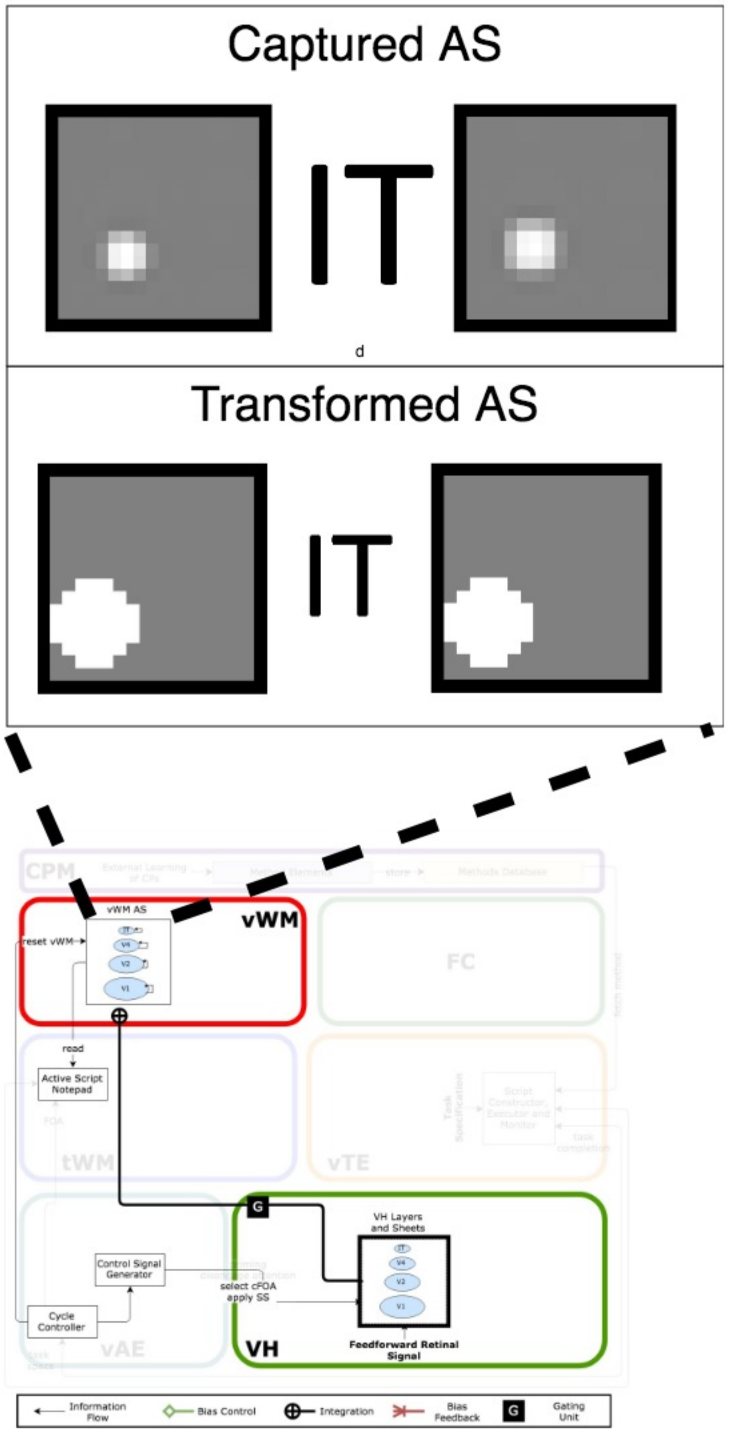


Figure B.7: The AS consisting of the feature maps (or sheets) in the *IT* layer are stored in the vWM using an Integration operation. The captured AS is transformed to represent location information only. See text for more information.

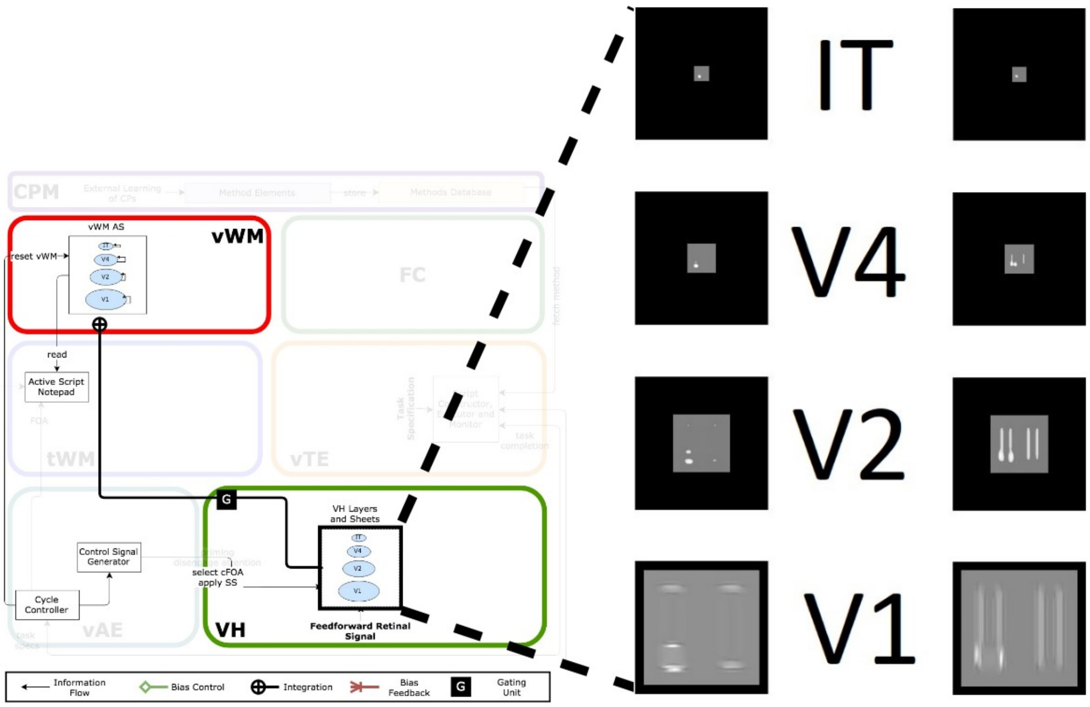


Figure B.8: Incoming target processed by the VH when no priming operation in place. Whiter areas indicate higher neural activations.

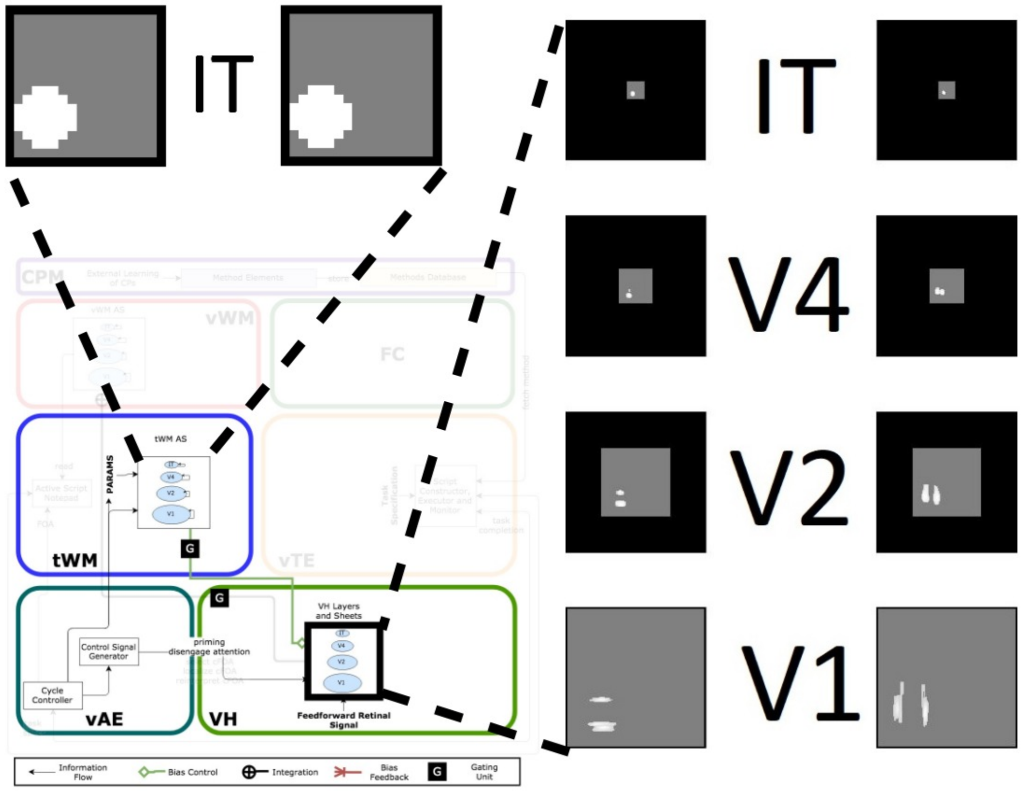


Figure B.9: Incoming target processed by the VH when priming operation in place. Whiter areas indicate higher neural activations.

of a correctly and incorrectly cued target. Notice that on a correctly cued target, the neural activations are significantly higher since neurons have not been suppressed due to the priming CP that was placed earlier.

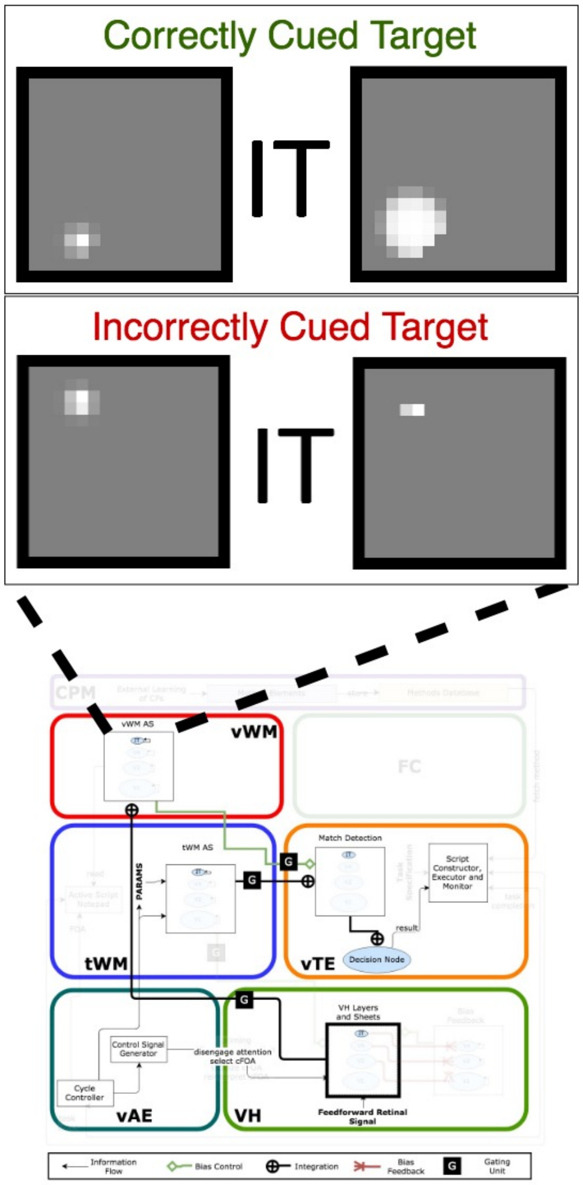


Figure B.10: During the presentation of a correctly or incorrectly cued target, contents are stored into the vWM. Whiter areas indicate higher levels of activation.

The contents of the tWM are then loaded with a prebuilt AS of a “square” using the command below.

```
BaseMethod::loadAStoTWM(square_as,
                              global_network->getNetwork("tWM_net"));
```

Following this, the feature detection process is initiated and the Match Detection Nodes output one of two results as shown Figure B.11. The system then waits for the result of this computation using equations 4.4 and 4.5 and outputs ***True*** or ***False*** for the correctly or incorrectly cued target respectively due to the differences in their neural activations.

```
detection = VAE()->VH()->Detection(FEATURE_DETECTION);
```

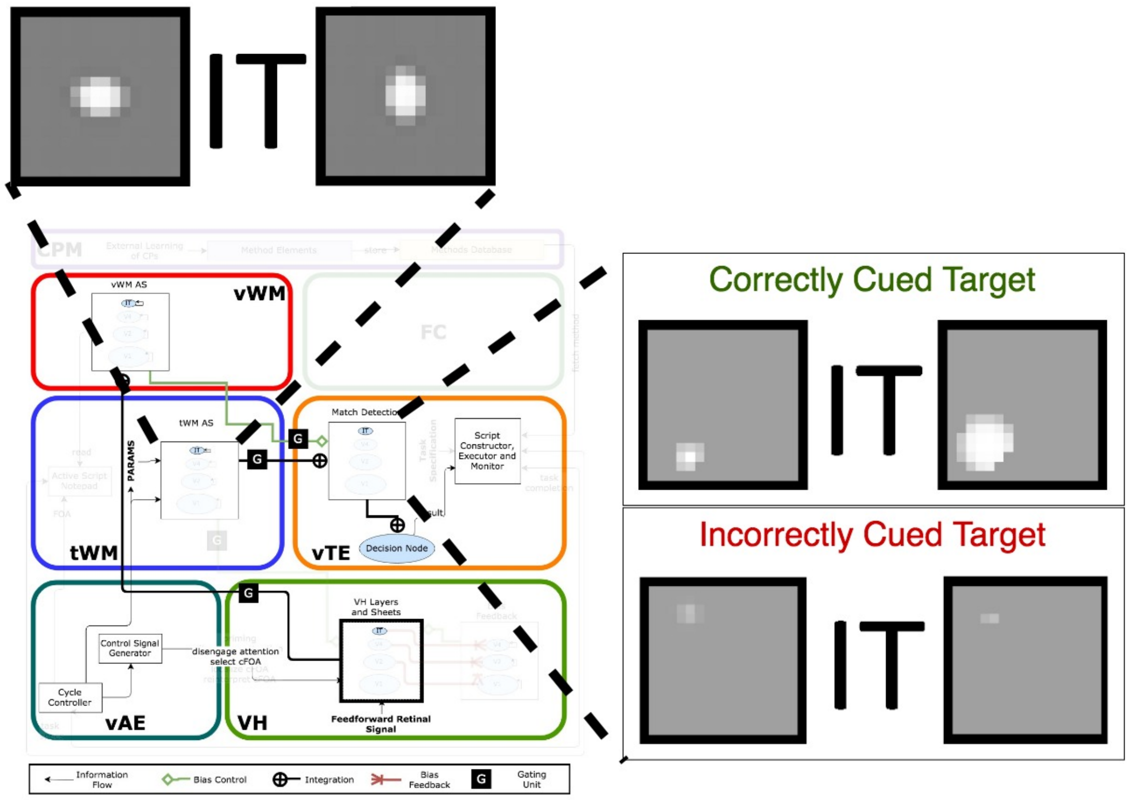


Figure B.11: The Match Detection Nodes receive information from the vWM and tWM. Used as part of the detection criteria. See text for more details.

Appendix C STAR Parameters

C.1 Visual Hierarchy Parameters

All of our images are 256x256 in size and the formula for the visual angle $V = 2arctan(\frac{S}{2D})$ was used to calculate the approximate size S in pixels of the stimuli within the images. The size of each layer was V1: 128x128, V2: 64x64 V4: 32x32, IT_{INT}: 16x16 and IT_{TOP}: 16x16.

The VH module represents the full ventral and dorsal streams of the visual processing areas implemented as a four layer network: V1->V2->V4->IT. The IT layer is divided into IT_{INT} and IT_{TOP}, the intermediate and top layer respectively. IT_{INT} is called as such since it is one of the intermediate layers of the VH and contains category information about the stimulus. On the other hand, IT_{TOP} is the top layer with respect to the VH processing feedforward input and is used as a selection layer necessary for the $\theta - WTA$ algorithm.

Each layer except for IT_{TOP} contains 8 feature sensitive Nodes for orientations: [*-90,-67.5,-45,-22.5,0,22.5,45,67.5*] and 3 feature sensitive Nodes for color: [*red, green, blue*] for a total of $8x3 = 24$ Nodes. Since IT_{TOP} is a selection layer, it contains only a single Node. To simulate orientation selectivity, a Gabor filter was used from the input stimulus to layer V1 while Gaussian filters were used for connecting layers V1 -> V2 -> V4 -> IT.

C.2 Experiment Parameters

Egly & Driver 1994

In this experiment, the fixation point was $0.1°x0.1°$ and the two rectangles of size $1.7°x11.4°$ with a stroke width of 0.2°, centered 4.8° from the fixation cross. The cue was represented by a 0.2° thickening of any of the corners of the rectangle. The target appeared at any of the corners of the rectangle as solid squares of size $1.7°x11.4°$. Using these parameters, the size of the fixation was $2x2$ px. The rectangle was $29x194$ px with a width of 4 px. The cue width was 4 px and the target $29x29$ px. During this detection experiment, the decision criteria was set in the range of $[0.1, 0.26]$. The cue and ISI were presented for 100 and 200ms. Target presentation occurred for 2,000 ms or until the subject responded with a button press and the fixation was presented for 1,000 ms.

Folk et al. 1992

In this experiment, each box was $1.15°x1.15°$ and the cue circles were $0.36°x0.36°$. The center to box distance was 4.7° and the targets presented within a box were 0.57°. Using these visual angles, and an image size of 256x256, the size of the box was $20x20$ px, the cue circles $6x6$

px, the center to box distance 80 px, and the target size $10x10$ px. The fixation display was presented for 1000 ms, while the cue, ISI and target displays were presented for 50 ms each.

Raymond et al. 1992

All stimuli were upper case letters spanning $0.82°x0.82°$. All stimuli including the probe was a green color while the white target was modeled as a red color. Each target appeared for a brief 15 ms period followed by a 75 ms interval where a blank white mask was shown. To allow neural activations to stabilize, 2 pre-target items were presented on each condition. The identity of the pre-target items did not influence the system output. Hence, the target was always the 3^{rd} letter in the stream.

When building partial attentional samples as templates to be stored into tWM, only the neural activations at the $IT_{INTERMEDIATE}$ layer are recorded since there is insufficient time for getting information at lower layers due to the short stimulus presentation time. These were collected in the same way a regular experiment would be run. Namely, the stimulus was presented, a selection allowed to occur at the top of the VH and the resulting information at $IT_{INTERMEDIATE}$ used to build an AS. This is then stored into one of the *slots* of tWM.

We decided to choose a subset of letters to present to the system based on their distinguishability. Hence, all stimuli were only the following letters: A,B,C,D,G,E,H,I,O,Q.

Spatial and Feature Attention

In the McAdams & Maunsell 1999 [40] paper, the fixation point was 0.7° visual angle, the spatial frequency was set to 1-5 cycles/degree and the size of the stimuli was $0.6° - 2.2°$. The authors noted that they had chosen the stimuli size and spatial frequency that resulted in maximal responses of the neuron being recorded. Using these parameters, we set the fixation point to be $12x12$ px, and the size of the stimuli to be $38x38$ px. The width of the orientation bar was adjusted within this range to yield a maximal response of the neuron with its preferred feature. The image size was $256x256$.

C.3 Complete STAR Architecture

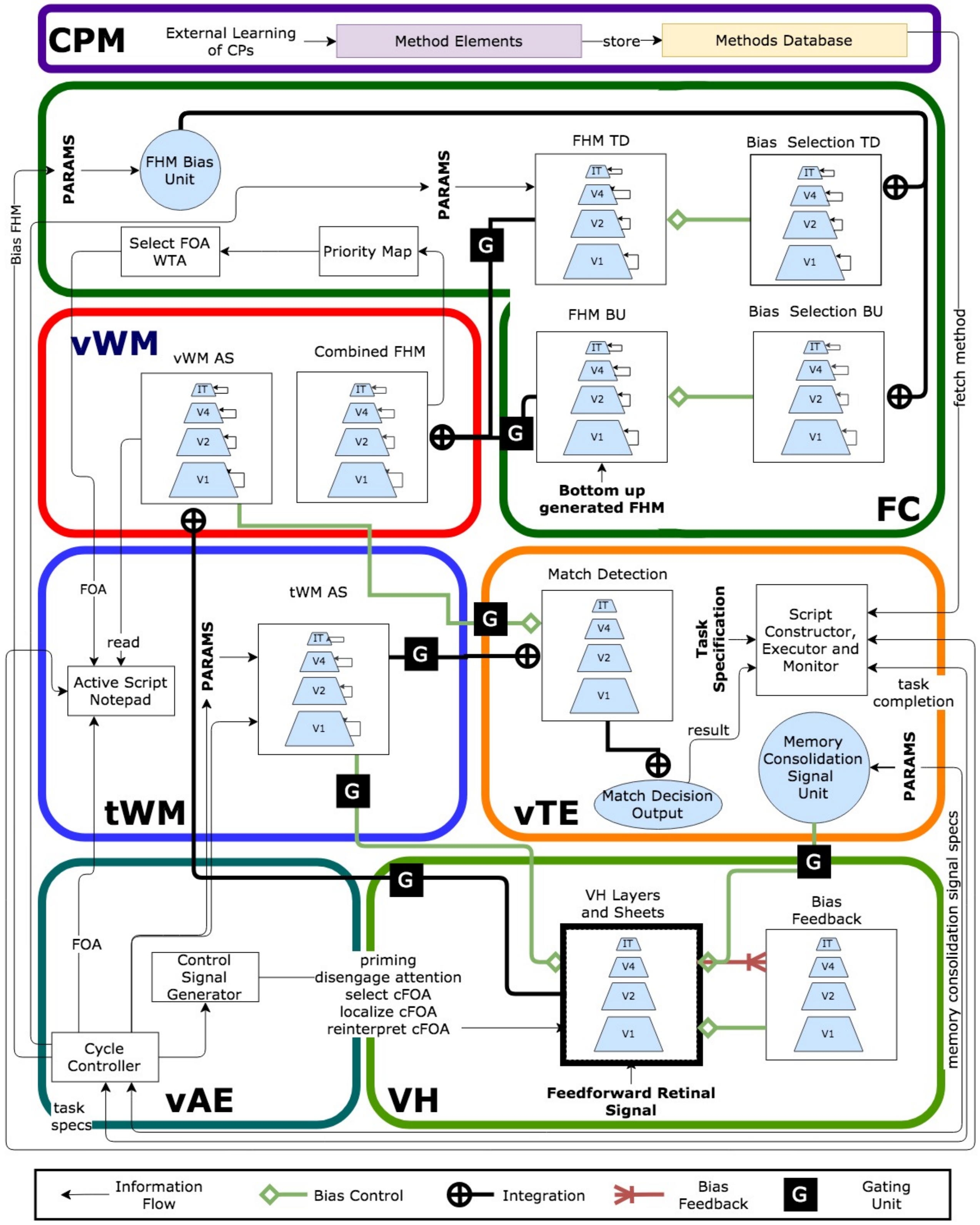


Figure C.1: Complete STAR Architecture