Biologically Plausible Neural Learning using Symmetric Predictive Estimators

by

David Xu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2016

© David Xu 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

A predictive estimator (PE) is a neural microcircuit hypothesized to explain how the brain processes certain types of information. They participate in a hierarchy, passing predictions to lower layers, which send back prediction errors. Meanwhile, the network learns optimal connection weights in order to minimize the prediction errors. This two-way process has been used to hypothesize models for brain mechanisms, such as visual information processing. However, the standard implementation for a PE uses the same weight matrix for both feed-forward and feed-back projections, which is not biologically plausible. In this thesis, we investigate the predictive estimator using individual feed-forward and feed-back connection weights. We extend the model and introduce the Symmetric Predictive Estimator (SPE). We investigate the dynamics of a SPE network, analyze its stability, and define a general learning rule. A functional model of the SPE is implemented analytically, and in a neural framework using spiking neurons. Both implementations are built as Python modules that can accept generic, numerical inputs. With a series of general experiments, we demonstrate its ability to learn non-linear functions and perform a supervised-learning task. This variation on the PE may provide insights to the theory of predictive estimators and their role in the brain.

Acknowledgements

I give my deepest thanks to my supervisor, Professor Jeff Orchard, for his guidance, patience and mentorship he has provided me over the past years. His passion for research, teaching and outlook on life has inspired me to make the best of my time in Waterloo. Having Jeff as a supervisor has undoubtedly had a positive impact on my life.

I would also like thank my committee members, Professor Gladimir Baranoski, and Professor Chris Eliasmith, for spending time to read and analyze my thesis. Their comments will provide me a new perspective on the research, and improve the quality of the thesis.

Much appreciation goes to the Centre for Theoretical Neuroscience (CTN) at the University of Waterloo. Their courses, research and presentations have provided incredible insight to my understanding of computational neuroscience.

I would like thank my friends in the Scientific Computation Group for their support and company, as well as my previous lab mates Andrew, Cameron and Lin. In addition, much gratitude goes to my girlfriend, Megan, for her patience and support.

Dedication

This is dedicated to my Grandma and my parents, for their sacrifices to provide me with opportunities they never had.

Table of Contents

List of Tables

List of Figures

List of Symbols

The following are commonly occurring symbols used throughout this thesis.

β	Network weighting parameter
v	State representation in the PE Model
ε	Feed-back residual error for PE or feed-forward residual error for SPE.
f	Feed-forward transformation function.
g	Feed-back transformation function.
γ	Learning rule time constant.
K	Learning rate.
A	Bottom-up input.
V	Top-down expectation.
τ	General time constant.
r	State representation in the SPE model.
δ	Feed-back residual error for SPE.
N	Number of basis functions for functional mapping.
ϕ	Basis function.
$\vec{b}$	Feed-forward function basis coefficient.
$\vec{c}$	Feed-back function basis coefficient.
E	Error for the functional mapping.
D	The domain of the functional mapping.
T	Total simulation time.
t	Simulation time.

dt	Simulation time-step.

Chapter 1

Introduction

Perception is an active process where sensory input and expectation are used to intelligently perceive one's surroundings [15]. An example of this two-way process is shown in Figure 1.1. The Necker cube is an optical illusion which provides ambiguity to the human visual system [29]. What the eyes see as a set of connected lines, the mind interprets as a cube in one of two different orientations. But what mechanisms in the brain result in these two different percepts? Another example is how humans may interpret patches of clouds in the sky differently. The same visual input of a cloud may register differently between people based on their prior knowledge of objects. How does one's prior knowledge factor into perception? How might the brain model a two-way process?

Modeling the dynamics of the brain falls into the field of computational neuroscience. This interdisciplinary field studies the brain function and structures. The focus is not to produce models that outperform human intelligence, but to create biologically realistic models that match the physiological structure and dynamics of biological data. There is a tremendous amount of research focused on the neuron which is considered the building block of the brain. Neurons interact with each other in the form of action potentials, an all-or-nothing spike of voltage between interconnected neurons. There are over 85 billion neurons present in the human brain [1]. This vast network of "on or off" neurons can be simulated in the form of spiking neural models, which hypothesize how the brain functions.

A predictive-estimator network harmonizes data given multiple inputs. Cortical circuits called predictive estimators (PEs) participate in a hierarchy of feed-forward and feed-back connections. Lower layers pass sensory input to higher layers, while higher layers pass predictions to lower layers. These PE units learn a set of connection weights that translate signals between the layers. PE models have been built that represent the visual system,

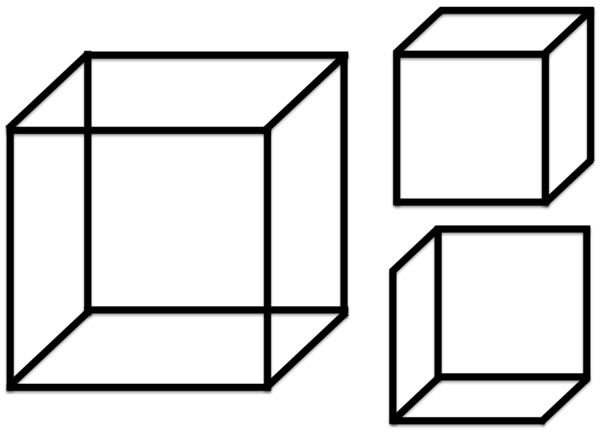


Figure 1.1: The Necker cube [29] can be seen with the bottom left face as the outwards face forming a cube (top) or the top right face can also be seen as the outwards face forming a similar cube (bottom).

in which the bottom layer (e.g. rods and cones) and top-down prediction from upper layers (e.g. cognition) form an internal state representation [30]. However, the standard implementation lacks biological plausibility since it uses the same weight matrix for feed-forward and feed-back projections. The process of weight copying is considered biologically implausible due to physical constraints. How could a set of connection weights be copied in the brain? What biochemical mechanism, operating within a given neuron, could duplicate the synaptic weight of a connection between two other neurons [16]? In addition, it is unclear if predictive estimators have been implemented using a spiking neural model, one that simulates the temporal spiking characteristics of each individual neuron.

In this thesis, we address the biological implausibility of copied connection weights, by investigating the development of a PE unit with individual feed-forward and feed-back connection weights. We introduce a variation on the model, called a symmetric predictive estimator (SPE), that can learn complex, non-linear transformations in a biologically plausible manner. Network connection weights are adjusted using an error signal that is local to the connection. There is no need for weight copying between feed-forward and feed-back connections. We also implement the model using the Neural Engineering Framework [12], which allows us to simulate individual neuron spikes for increased biological plausibility.

This thesis is organized into several chapters that cover topics about the PE and SPE. Chapter 2 provides an overview of PEs and their usage in neuroscience theory. The motivation and biological plausibility of the PE are covered. We also discuss the differences between the SPE and other existing theories. Chapter 3 describes the theory and dynamics of a PE implementation with individual feed-forward and feed-back weights. Chapter 4 introduces a variant of the PE model, called the SPE, and the dynamics of the SPE model are investigated. Chapter 5 details the implementation of the SPE model in an analytical simulation, and using a neural framework with spiking neurons. Chapter 6 proposes a series of generic experiments to test the dynamics of the SPE. Details of each simulation are given, and the results are evaluated. In addition, we compare the results between the analytical and spiking neuron implementation. Chapter 7 concludes the thesis with comments and contributions of the SPE. We discuss obstacles with the dynamics of larger SPE models, and outline the future work.

Chapter 2

Background and Related Work

2.1 Predictive Coding in Neuroscience

Predictive coding involves the use of predictive estimators connected in a hierarchy, passing predictions to lower layers, which in turn pass back prediction errors to upper layers. Bastos et al. [2] provide a thorough overview of predictive coding. They highlight electrophysiological experiments that are consistent with the predictive coding theory. They find that the microcircuitry in the cortical columns are similar to the structural connections in predictive coding. Cortical columns are groups of vertically connected neurons that express similar receptor field properties. For example, they share a correlated response to a set of stimulus, such as visual orientation. Bastos et al. [2] formulate a canonical microcircuit that is suitable for predictive coding, and is consistent with the theory of cortical columns. This shows that variations of predictive estimators have been formulated to explain structural functions of the brain.

Feed-back connections were traditionally thought to be weak and modulatory in the sense they could not elicit spikes in the post-synaptic neuron. They were observed to only modulate certain transmission aspects in the post-synaptic neuron [35]. Compare this to feed-forward connections that are stronger, and can elicit spiking in their post-synaptic neurons [2] [35]. However, recent studies have shown that feed-back connections can also elicit spikes in the same way feed-forward connections can [8] [9] [27]. This suggests exploring functional symmetry on the predictive estimator may provide further insights on predictive coding.

Predictive estimators have also been used to hypothesize a mechanism for the *mirror neuron system* [22]. The *mirror neuron system* has been observed in macaque monkeys and is attributed to a link between action execution and action observation [31]. Studies showed that certain neurons in a monkey would activate when it observed humans picking up a piece of food. However, these same neurons would activate when the same monkey picked up the food [11] themselves. These neurons were dubbed as *"mirror neurons"*, because the subject's observation of an action produced mirrored neural activity as if the same subject performed the action themselves.

Kilner et al. [22] has suggested that the mirror neuron system can be implemented using the predictive coding framework. Jacob et al. [19] suggest the *mirror neuron system* does not account for the intentions of an action. They provide a thought experiment involving a surgeon in an operating room. When the surgeon is grasping a sharp scalpel, how can the *mirror neuron system* differentiate between the intention to cure or the intention to hurt the patient? The observer may mirror neural activity that correspond to the surgeons motor actions, but what accounts for the intentions of the action? There should be differences in perception between the intention to cure and intention to harm, but the differentiation is not possible in a purely feed-forward system using only visual information.

Kilner et al. [22] argue that the predictive estimator framework supports a feed-back signal that provides information regarding the context of the action. This allows for mirrored neural activity at the kinematic level, but different activity at the contextual level. They find that predictive estimators can account for the feed-back signal. However no functional implementation of a predictive estimator model has been provided. We hope that the work in this thesis may help provide a generic, functional model to help support existing theories.

The Kalman filter is a popular mechanism used to estimate real-world values that have a degree of uncertainty [21]. It is widely used in signal processing and is often applied in navigation systems. A common example is determining the precise location of a vehicle using the physics of motion and external measurements from a GPS [23] [24]. The vehicle's position can be predicted by integrating its velocity over time from its throttle control to determine the distance travelled, and adding it to its previous position. However this technique alone would be inaccurate over time because the integration does not account for external influences on the vehicle. Conversely, a GPS device can provide a measurement based on triangulated antenna signal strengths, but it still offers a level of uncertainty. The Kalman filter is able use both the prediction, and a measurement to form a new estimate of the location at the next point in time. The theory of the Kalman filter has been used to model brain processes and has been investigated in the Neural Engineering Framework [12]. It also has been used in neuroscience to replicate a motor control system [10].

The Kalman filter uses a weighted average of the prediction and measurement to form an estimate. The weighted average is calculated based on the covariance of its prediction and measurement. This method is encapsulated into a parameter called the Kalman gain [21]. This parameter weights the relative influence of the prediction and measurement on the next calculated state [23]. This draws a similar parallel to the gain parameter, β, that is used in the SPE model introduced in this thesis. Both the Kalman filter, and SPE model use a gain parameter in its calculations of the next state value. However, the primary goal of the Kalman filter is to form a time-dependent estimate on the next state. The error on the estimate is used to change the Kalman gain to form a better estimate on the next iteration. Contrast this to the goal of our model, where the SPE is used to form a hierarchy of multiple state representations, with each level of the hierarchy having a different interpretation of the input values. An example given in this thesis is Cartesian co-ordinates in one level predicting the corresponding polar co-ordinates in the next level of the hierarchy. In addition, the prediction error in the SPE is used to learn connection weights to translate the state values between each level of the hierarchy. Currently, the prediction error is not used to affect the gain parameter.

Rao and Ballard introduced the usage of predictive estimators to functionally model some effects seen in the visual cortex [30], the region of the brain responsible for processing visual information. Their hierarchical model utilizes a feed-back signal that carries predictions about the input neural activity, and a feed-forward residual error signal. This residual error is the difference between the prediction and the actual neural activity. It is important to note that this is not a prediction in the sense of a future point in time, but rather the effect from an alternative source of information or interpretation of the input [22].

They designed an experiment and provided a mechanism to explain visual phenomena observed in the neurons of the visual cortex. Their experiment and method are explained in more detail in “Appendix B”. They concluded that the behavior of certain neurons in the visual cortex are not only driven by a feed-forward sensory input, but are also influenced by a feed-back prediction. Their network of PEs was able to match observations from biological data. They suggest the PE network might be a mechanism that neurons use in the visual cortex.

However, their model uses feed-forward and feed-back connection weights that are matrix transposes of each other. The use of copied connection weights may decrease the biological plausibility of the system [5]. This is because the weight changes from a neural connection need to be instantly transported to the copied weight connection. The physical constraints of this process are implausible in the brain [16]. In this thesis we build on the PE theory and experiment with individual connection weights for the feed-forward and feed-back connections. The final result is a Symmetric Predictive Estimator (SPE) which we test analytically and implement using spiking neurons.

In this thesis, we provide an analytical model of the SPE network which is implemented using numerical methods (Euler stepping). We also implement the model using the Neural Engineering Framework (NEF), which is a framework that can construct and simulate neural systems. An outline of the NEF is given in “Appendix A”.

2.2 Neural Learning

Artificial neural networks are often used to perform supervised-learning tasks on hard-to-solve problems such as image classification [6]. The design of artificial neural networks is inspired by the brain’s vast inter-connectivity of neurons. Abstract nodes are used to mimic the neuron, and nodes can elicit responses in other nodes by modelling the synapse firing between neurons [20]. Connection weights exist between nodes to communicate information between them. These connection weights are initially random, and a common algorithm used to change connection weights between nodes is called backpropagation [33]. Nodes are grouped into an input layer, hidden layer and output layer.

Backpropagation uses a two-phase system to perform a supervised-learning task. First, the input is fed through the network starting from the input layer, which activates hidden layer nodes and subsequently output layer nodes. The activation at the output layer is the result of driving the input through feed-forward connections of the input layer, hidden layer and then the output layer. An error value is calculated as the difference between the desired output and the network’s output layer [20]. The second phase consists of updating these connection weights between the input-to-hidden and hidden-to-output layers. This is done using gradient descent, calculating the error gradients at each layer using the chain rule to apply a weight update. This method is highly effective in machine learning tasks but many argue this is not what the brain does [25] [16].

Bengio et al. [5] suggest reasons why backpropagation is not biologically plausible. Mainly, the error gradient that is used to calculate each weight is derived from the output layer and transported instantaneously to each node. This would require the network of neurons to have a global access point to the output error values, which seems implausible in a physical system. In addition, there are no feed-back connections in the backpropagation algorithm, but these connections are observed in brains [7]. If feed-back connections were used to project the error back through the network, they would require transposed weights of the feed-forward connections, which requires copying connection weights.

In this thesis, we demonstrate the SPE’s ability to perform a supervised-learning task using feed-back connections that provide a local error for learning. The error nodes are inherent to the dynamics of the SPE model and are not provided by an external value. In addition, the SPE can be implemented using spiking neurons. This is done using the NEF and we demonstrate that it can still perform a supervised-learning task. Implementation into the NEF also introduces temporal delays to the network, thus, addresses the implausibility of instantaneous transport.

Lee et al. [25] address the biological plausibility of backpropagation with a method called difference target propagation (DTP). They introduce a learned feed-back function between layers. The desired value (target) is propagated backwards using feed-back connections and this allows the error to be calculated locally in-between layers. Their implementation provides results comparable to backpropagation. However, there are key differences between the SPE model and DTP.

First, the goal of the SPE model is to incorporate both bottom-up and top-down input to form a perception. Contrast this to the goal of backpropagation and DTP, where the top-down connections are only used to train the feed-forward weights. In a fully trained DTP network, values at each layer are driven purely from a feed-forward standpoint. However, our model incorporates a β term which can weight the relative influence of the feed-forward or feed-back signals on perception.

Secondly, the DTP algorithm appears difficult to implement in a continuous time domain, and in a neural framework. The algorithm requires a difference calculation using the same feed-back connection at two points: the hidden layer's state, and the target. If implemented in a framework such as the NEF, two nodes with the same connection weights would be required to calculate this value. This would require weight copying in order to evaluate the feed-back function for two different inputs at the same time, which seems implausible in a physical system [16]. Alternatively, time-slicing could be applied to evaluate two points with the same connection, but that would also require additional time synchrony and memory. The SPE model differs because it is derived as a system of dynamics equations, and can be implemented in a neural framework. In addition, the SPE does not use a difference in feed-back propagation to calculate the target.

In this thesis, we outline the development of the SPE and the dynamics of a SPE network. We do not attempt to replicate the experiments mentioned in this chapter. Instead we test the SPE network using a simplified set of inputs and top-down predictions as an initial demonstration of the network.

Chapter 3

Predictive Estimators

3.1 Model

Rao and Ballard introduced a predictive-estimator model for visual processing [30]. Their model consists of a feed-back signal which carries prediction about the input, and a feed-forward signal which carries the residual error between the input and prediction.

The work presented in this thesis is inspired by the hierarchical model presented in Rao and Ballard's paper [30] and expands on their theory by introducing individual feed-forward and feed-back transformations. The predictive-estimator (PE) unit consists of two main components: 1) v_i: a state representation and 2) ε_i: residual error between v_i and the prediction from a higher level v_{i+1}. Between PE levels there is a feed-forward connection, f_i, which conveys the residual error coming from a lower level, and a feed-back connection, g_i, that delivers the prediction from a higher level. Figure 3.1 shows a network diagram of two PE units connected together. This model introduces individual feed-forward and feed-back connections in the PE. The connection weights of f_i are not the transpose of g_i. The dynamics of the PE unit are governed by the differential equations

$$\tau_v \frac{dv_i}{dt} = (1-\beta) f_{i-1}(\varepsilon_{i-1}) - \beta \varepsilon_i, \tag{3.1}$$

$$\tau_\varepsilon \frac{d\varepsilon_i}{dt} = v_i - g_i(v_{i+1}) - \varepsilon_i, \tag{3.2}$$

where τ represents a time constant, and β acts as a tuning parameter that weights the feed-forward and feed-back connections. A β value of 0.5 would drive the network in a balanced state. This parameter can be tuned to drive the network in a purely feed-forward

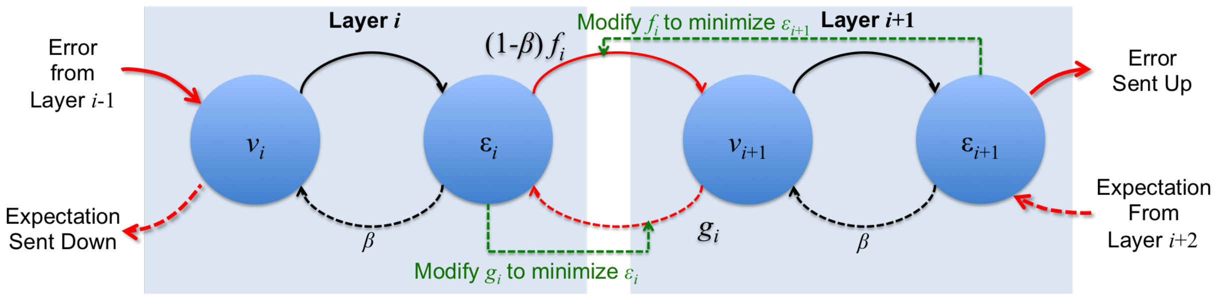


Figure 3.1: Two Predictive-Estimator (PE) Units connected together. Each blue circle is a node representing the value of the respective variable enclosed. The v node represents a state representation, and the ε node represents the residual error. The layer index is represented by the subscript i. A solid black arrow represents addition of the tail-end value to the arrow-end node, while a dotted black arrow represents subtraction. A solid red arrow implies the tail-end value undergoes a transformation function and is added to the arrow-end node, while a dotted red arrow implies subtraction. A dotted green arrow implies a modulating connection, in which connection weights are modified based on the tail-end node's value. Note that the recurrent connection from ε to itself, as describedd in (3.2), has been omitted from the figure for simplicity.

manner, by setting $\beta = 0$. This implies there is zero weighting on feed-back connections. Conversely if $\beta = 1$ there is zero weighting on feed-forward connections. The function f_i models how the residual-error will affect a higher-level state, while g_i represents how the prediction affects a lower-level state. The f_i and g_i functions may be pre-defined; however, a characteristic of this PE is the ability for the network to learn an internal representation of feed-forward and feed-back functions that formulate a state representation v_i.

3.2 Learning Rule

The PE modifies f_i and g_i connections in order to minimize the error ε_i between units. Learning rules for f_i and g_i in a one-dimensional linear case are

$$\gamma_f \frac{df_i}{dt} = -K\varepsilon_i\varepsilon_{i+1}, \tag{3.3}$$

$$\gamma_g \frac{dg_i}{dt} = K v_{i+1}\varepsilon_i, \tag{3.4}$$

where γ represents a time constant, and K represents a learning rate. The PE model will shape the state representation v_i such that ε_i is as close to zero as possible. If $\varepsilon_i = 0$ then (3.3) and (3.4) will evaluate to zero, and there will be no changes to the feed-forward and feed-back mappings. If $\varepsilon_i \neq 0$, (3.3) and (3.4) will be non-zero, and change the feed-forward and feed-back mappings in order to find a minimum ε_i for the system.

3.3 Stability Analysis

Although this model provided a stable system with linear feed-forward and feed-back mappings, simulations showed that this system was not stable for non-linear cases. It would be ideal that the PE is stable for non-linear mappings because many processes are non-linear in nature. A generic non-linear mapping is taking Cartesian co-ordinates (x, y), and mapping them to polar co-ordinates (ρ, θ), and *vice versa*. Figure 3.2 shows a PE network that models this system. The input to the PE would be x and y co-ordinates, while the top-down expectation would be the matching ρ and θ values. There is no corresponding v_0 state representation because the residual error ε_0 can be calculated directly between the input (A_x, A_y) and prediction $g(V_\rho, V_\theta)$.

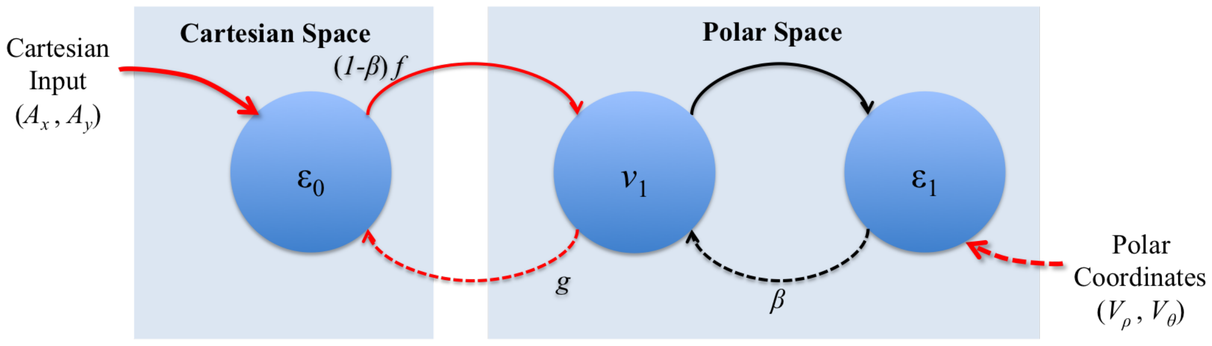


Figure 3.2: Cartesian and Polar PE Units connected. Cartesian co-ordinates A_x and A_y, are input to the left PE unit and Polar co-ordinates V_ρ and V_θ are the predictions for the right PE unit.

For simplicity, the system to be analyzed has fixed f and g transformations, where f is a linear mapping for the error, and g is the transformation from polar co-ordinates to

Cartesian co-ordinates such that

$$f(\varepsilon_0) = \begin{bmatrix} f_s \varepsilon_{0x} \\ f_s \varepsilon_{0y} \end{bmatrix}, \tag{3.5}$$

$$g(v_1) = \begin{bmatrix} v_{1\rho} \cos v_{1\theta} \\ v_{1\rho} \sin v_{1\theta} \end{bmatrix}. \tag{3.6}$$

After running simulations it became apparent that the network is not stable for this mapping, even when the f and g mappings are fixed and not changed by (3.3) and (3.4). We can investigate the stability of this system by analyzing the network. The dynamics of this system are governed by the differential equations

$$\tau_\varepsilon \frac{d\varepsilon_{0x}}{dt} = A_x - v_{1\rho} \cos v_{1\theta} - \varepsilon_0 x, \tag{3.7}$$

$$\tau_\varepsilon \frac{d\varepsilon_{0y}}{dt} = A_y - v_{1\rho} \sin v_{1\theta} - \varepsilon_0 y, \tag{3.8}$$

$$\tau_v \frac{dv_{1\rho}}{dt} = (1-\beta) f_s(\varepsilon_{0x}) - \beta \varepsilon_{1\rho}, \tag{3.9}$$

$$\tau_v \frac{dv_{1\theta}}{dt} = (1-\beta) f_s(\varepsilon_{0y}) - \beta \varepsilon_{1\theta}, \tag{3.10}$$

$$\tau_\varepsilon \frac{d\varepsilon_{1\rho}}{dt} = v_{1\rho} - V_\rho - \varepsilon_{1\rho}, \tag{3.11}$$

$$\tau_\varepsilon \frac{d\varepsilon_{1\theta}}{dt} = v_{1\theta} - V_\theta - \varepsilon_{1\theta}, \tag{3.12}$$

where the differential equations for ε_0, v_1 and ε_1 are split component-wise, A_x and A_y represent the components of the input to the PE units, while the corresponding V_ρ and V_θ represent the top-down input.

Let $\vec{S}$ represent state variables in the system where

$$\vec{S} = [\varepsilon_{0x}, \varepsilon_{0y}, v_{1\rho}, v_{1\theta}, \varepsilon_{1\rho}, \varepsilon_{1\theta}]^T. \tag{3.13}$$

Given an equilibrium point $\vec{S}^*$, we can linearize the system around that point, and inspect the sign of the real part of the eigenvalues to analyze the stability. Let our system of differential equations be written as

$$\tau \frac{d\vec{S}}{dt} = F(\vec{S}), \tag{3.14}$$

where F is the vector function that evaluates the right-hand sides of all the differential equations above. Given the equilibrium point $\vec{S}^*$, we can write $\vec{S}$ in terms of deviations from $\vec{S}^*$,

$$\vec{S} = \vec{S}^* + \Delta\vec{S}. \tag{3.15}$$

Thus, $\mathrm{F}(\vec{S}) = \mathrm{F}(\vec{S}^* + \Delta\vec{S})$, and we can write the Taylor expansion of F about $\vec{S}^*$ as

$$F(\vec{S}^* + \Delta\vec{S}) = F(\vec{S}^*) + \nabla F(\vec{S}^*)\Delta\vec{S} + \mathcal{O}(\Delta\vec{S}^2). \tag{3.16}$$

The stability of the system can be analyzed by using a linear approximation of $\mathrm{F}(\vec{S})$. The first term will equate to zero because it is the equilibrium point, while we can ignore the third term because higher order terms are negligible with a small step. This simplifies to

$$F(\vec{S}) \approx \nabla F(\vec{S}^*)\Delta\vec{S}, \tag{3.17}$$

which is a linear function of $\Delta\vec{S}$. Thus, the dynamics of the system near the equilibrium can be approximated by

$$\tau\frac{d\Delta\vec{S}}{dt} = \begin{bmatrix}\Delta\dot{\varepsilon}_{0x}\\ \Delta\dot{\varepsilon}_{0y}\\ \Delta\dot{v}_{1\rho}\\ \Delta\dot{v}_{1\theta}\\ \Delta\dot{\varepsilon}_{1\rho}\\ \Delta\dot{\varepsilon}_{1\theta}\end{bmatrix} = \begin{bmatrix} -1 & 0 & -\cos v_{1\theta} & v_{1\rho}\sin v_{1\theta} & 0 & 0\\ 0 & -1 & -\sin v_{1\theta} & -v_{1\rho}\cos v_{1\theta} & 0 & 0\\ (1-\beta)f_s & 0 & 0 & 0 & -\beta & 0\\ 0 & (1-\beta)f_s & 0 & 0 & 0 & -\beta\\ 0 & 0 & 1 & 0 & -1 & 0\\ 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}\begin{bmatrix}\Delta\varepsilon_{0x}\\ \Delta\varepsilon_{0y}\\ \Delta v_{1\rho}\\ \Delta v_{1\theta}\\ \Delta\varepsilon_{1\rho}\\ \Delta\varepsilon_{1\theta}\end{bmatrix}. \tag{3.18}$$

Suppose we set β=$\frac{1}{2}$ and f_s=1, and choose some input $(V_\rho,\ V_\theta) = (2,\ V_\theta)$, and set $(A_x,\ A_y)$ to be the corresponding Cartesian input. It can be shown that $\vec{S_1}$ is an equilibrium solution where

$$\begin{aligned}\vec{S_1} &= [\varepsilon_{0x}, \varepsilon_{0y}, v_{1\rho}, v_{1\theta}, \varepsilon_{1\rho}, \varepsilon_{1\theta}]^T && (3.19)\\ &= [0\ \ ,0\ \ ,2\ \ ,V_\theta\ ,0\ \ ,0\ \ ]^T. && (3.20)\end{aligned}$$

At that equilibrium solution, the eigenvalues of the system matrix in (3.18) can be written in terms of V_θ,

$$\begin{bmatrix} -\frac{1}{2} + \frac{1}{2}\sqrt{-1 - 3\cos V_\theta + \sqrt{9\cos V_\theta{}^2 - 8}}\\ -\frac{1}{2} - \frac{1}{2}\sqrt{-1 - 3\cos V_\theta + \sqrt{9\cos V_\theta{}^2 - 8}}\\ -\frac{1}{2} + \frac{1}{2}\sqrt{-1 - 3\cos V_\theta + \sqrt{9\cos V_\theta{}^2 - 8}}\\ -\frac{1}{2} - \frac{1}{2}\sqrt{-1 - 3\cos V_\theta + \sqrt{9\cos V_\theta{}^2 - 8}}\\ -1\\ -1 \end{bmatrix}. \tag{3.21}$$

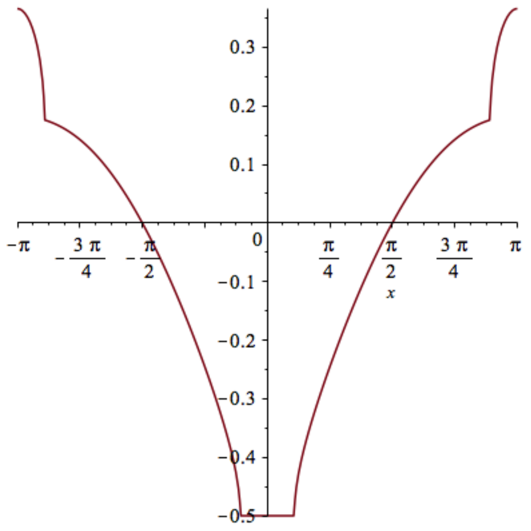


Figure 3.3: Instability of the PE system. Plot of equation (3.22) with respect to V_θ on the x-axis, and the real-part eigenvalue on the y-axis.

It can be shown that the real part of the eigenvalues are negative except the matching eigenvalue terms in rows 1 and 3

$$-\frac{1}{2}+\frac{1}{2}\sqrt{-1-3\cos V_\theta+\sqrt{9\cos {V_\theta}^2-8}}. \tag{3.22}$$

The eigenvalue for a given V_θ becomes positive in a certain range, as shown in Figure 3.3. At least one eigenvalue with a positive real part indicates the system would be pushed away from an equilibrium point given a perturbation from a steady-state solution [28]. For $\vec{S_1}$, a V_θ value that falls outside of $\pm\frac{\pi}{2}$ would yield eigenvalues with a positive real part. Hence, this network would not settle into a state in which the error node encodes a small value. This PE network is unable to fully cover the non-linear transform of polar-to-Cartesian co-ordinates.

This led to the design of the symmetric predictive estimator (SPE), which maintains the stability, individual feed-forward and feed-back weights, local error connections and is able to learn more complex transformations.

Chapter 4

Symmetric Predictive Estimators

4.1 Model

The SPE unit is similar to the PE unit, but contains two error nodes instead of one. The main difference is that the state representation is sent up in a feed-forward manner rather than residual error. This forms a symmetry between the feed-forward and feed-back connections. An outline of a SPE network is given in Figure 4.1. Each SPE unit consists of 3 nodes: r_i the state representation, ε_i the error between the bottom-up signal and the state representation, and δ_i the error between top-down prediction and the state representation. If the state representation r_i matches the input from below and above, then the error nodes ε_i and δ_i will be zero.

The equations that govern the dynamics of the SPE unit are

$$\tau_r \frac{dr_i}{dt} = (1-\beta)\varepsilon_i - \beta\delta_i, \tag{4.1}$$

$$\tau_\varepsilon \frac{d\varepsilon_i}{dt} = f_{i-1}(r_{i-1}) - r_i - \varepsilon_i, \tag{4.2}$$

$$\tau_\delta \frac{d\delta_i}{dt} = r_i - g_{i+1}(r_{i+1}) - \delta_i, \tag{4.3}$$

where τ represents a time constant, and β acts as tuning parameter for feed-forward and feed-back weights in a similar manner to the PE model described in 3.1.

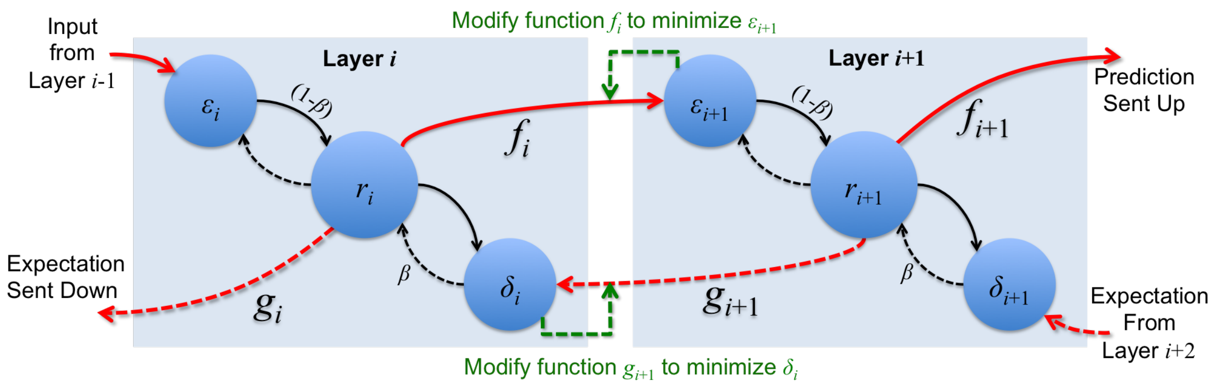


Figure 4.1: Two Symmetric Predictive Estimator (SPE) Units connected. The ε node represents the feed-forward residual error, r represents the state representation, and δ represents the feed-back residual error. Note that the recurrent connections from ε to itself, and δ to itself, as described in (4.2) and (4.3), have been omitted from the figure for simplicity.

4.2 Feed-forward and Feed-back Mappings

Ideally we would like the ability for the SPE to perform non-linear transformations. These transformations can be modeled using a linear combination of basis functions. Let the feed-forward transformation be defined as

$$f(r) = \sum_{n=1}^{N} b_n \phi_n(r), \tag{4.4}$$

where r represents the input, ϕ_n represents a basis function such as a linear interpolator or a Gaussian function, b_n represents the basis coefficient and n represents an index from a discrete set of N basis functions and coefficients. The intuition is that a transformation $f(r)$ can be represented by a linear combination of N basis functions and b_n coefficients. The feed-back transformation would be defined in a similar manner, but using coefficients labelled as c_n

$$g(r) = \sum_{n=1}^{N} c_n \phi_n(r). \tag{4.5}$$

4.3 Generalized Learning Rule

The SPE requires a generalized learning rule for the f and g functions. The linear combinations of basis functions, along with the delta rule [32] can be used for the development of a general learning rule. Let $\vec{b}$ define the coefficients for the basis functions such that

$$\vec{b} = (b_1, b_2, \ldots, b_N). \tag{4.6}$$

The equation for the feed-forward mapping (4.4) can be represented in vector form as

$$f(r_i) = \vec{\phi}(r_i)^T \vec{b}. \tag{4.7}$$

Consider the change in f_i in Figure 4.1. The variable ε_{i+1} is governed by

$$\tau_\varepsilon \frac{d\varepsilon_{i+1}}{dt} = f_i(r_i) - r_{i+1} - \varepsilon_{i+1}. \tag{4.8}$$

At steady state the change in ε_{i+1} should be zero such that

$$\frac{d\varepsilon_{i+1}}{dt} = 0, \tag{4.9}$$

and the value of ε_{i+1} at equilibrium is

$$\begin{aligned} \varepsilon_{i+1} &= f_i(r_i) - r_{i+1} && (4.10)\\ &= \vec{\phi}(r_i)^T \vec{b} - r_{i+1}. && (4.11) \end{aligned}$$

The error for the feed-forward transformation at steady state can be defined by

$$E(\vec{b}) = \|\varepsilon_{i+1}\|^2, \tag{4.12}$$

where error is a function of $\vec{b}$ because the basis functions do not change over time, but the co-efficients do. The learning will facilitate changes in weight coefficients $\vec{b}$. By substituting (4.11) into (4.12) we can perform gradient descent on the resulting equation where $\vec{\phi}(r_i)$ $= \vec{\Phi}$ for the sake of clarity,

$$
\begin{aligned}
\frac{\partial}{\partial \vec{b}} E(\vec{b}) &= \frac{\partial}{\partial \vec{b}} \left\| \vec{\Phi}^T \vec{b} - r_{i+1} \right\|^2 \\
&= \frac{\partial}{\partial \vec{b}} \Big[(\vec{\Phi}^T \vec{b} - r_{i+1})^T (\vec{\Phi}^T \vec{b} - r_{i+1}) \Big] \\
&= \frac{\partial}{\partial \vec{b}} \Big[(\vec{b}^T \vec{\Phi} - r_{i+1}^T)(\vec{\Phi}^T \vec{b} - r_{i+1}) \Big] \\
&= \frac{\partial}{\partial \vec{b}} \Big[\vec{b}^T \vec{\Phi} \vec{\Phi}^T \vec{b} - \vec{b}^T \vec{\Phi} r_{i+1} - r_{i+1}^T \vec{\Phi}^T \vec{b} + r_{i+1}^T r_{i+1} \Big] \\
&= \frac{\partial}{\partial \vec{b}} \Big[\vec{b}^T \vec{\Phi} \vec{\Phi}^T \vec{b} - 2 \vec{b}^T \vec{\Phi} r_{i+1} + r_{i+1}^T r_{i+1} \Big] \\
&= 2 \vec{\Phi} \vec{\Phi}^T \vec{b} - 2 \vec{\Phi} r_{i+1} \\
&= 2 \vec{\Phi} (\vec{\Phi}^T \vec{b} - r_{i+1}) \\
&= 2 \vec{\Phi} \varepsilon_{i+1}.
\end{aligned} \tag{4.13}
$$

Equation (4.13) shows that the change in coefficients $\vec{b}$ will be governed by the lower level state representation r_i and the residual error between lower and upper level, ε_{i+1}. Thus the learning rule for a feed-forward mapping is

$$\gamma_b \frac{d\vec{b}}{dt} = -K \varepsilon_{i+1} \phi(r_i), \tag{4.14}$$

where γ_b represents a time constant, and K acts as learning rate. Notice that if the error node from above, $\varepsilon_{\mathrm{i+1}}$, is zero, no changes will occur to $\vec{b}$, because there is no error in the feed-forward mapping. Conversely the learning rule for a feed-back mapping is

$$\gamma_c \frac{d\vec{c}}{dt} = K \delta_i \phi(r_{i+1}), \tag{4.15}$$

where $\vec{c}$ represents the coefficients for a feed-back connection g_{i+1}. Note that if δ_i is zero, no changes will occur to $\vec{c}$, because there is no error in the feed-back mapping. The similarity between learning rules (4.14) and (4.15) is expected due to the symmetry of the SPE unit.

These general learning rules can be applied to multi-dimensional transformations by having a set of independent f and g functions for each dimension the SPE unit is connected to. In addition, any error based learning rule can be implemented with this system because the error nodes δ and ε are a part of the SPE structure. For instance, our analytical model uses a direct implementation of this learning rule, while our NEF implementation uses a similar method described in Section 5.2.2.

4.4 Stability Analysis

The Cartesian and polar co-ordinates example from Section 3.3 can be revisited using the SPE. A network diagram is given in Figure 4.2. The lower-level SPE represents the Cartesian space where Cartesian co-ordinates are input into the lower-level ε_l node. The upper-level SPE represents the polar space, where polar co-ordinates are input into the higher-level δ_u node.

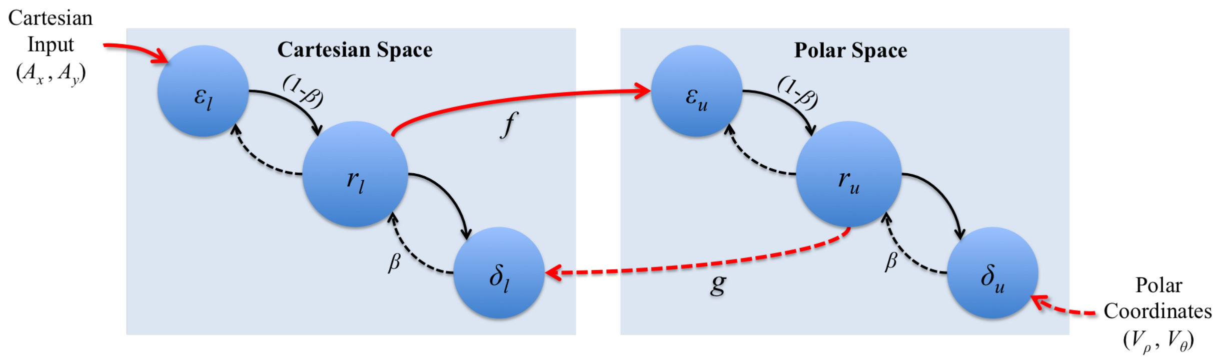


Figure 4.2: Cartesian and Polar SPE Units connected together.

Let the lower layer be represented by the subscript l and the upper layer be u, and the feed-forward mapping represented by f and the feed-back mapping represented by g. The function f is set to the Cartesian-to-polar transform, while g is set to the polar-to-Cartesian transform defined as

$$f(\vec{r_l}) = \begin{bmatrix} \sqrt{r_{lx}^2 + r_{ly}^2} \\ \arctan \frac{r_{ly}}{r_{lx}} \end{bmatrix}, \tag{4.16}$$

$$g(\vec{r_u}) = \begin{bmatrix} r_{u\rho} \cos r_{u\theta} \\ r_{u\rho} \sin r_{u\theta} \end{bmatrix}. \tag{4.17}$$

Based on Figure 4.2 the dynamics of the two SPE system is governed by equations

$$\tau_\varepsilon \frac{d\varepsilon_{lx}}{dt} = A_x - r_{lx} - \varepsilon_{lx}, \tag{4.18}$$

$$\tau_\varepsilon \frac{d\varepsilon_{ly}}{dt} = A_y - r_y - \varepsilon_{ly}, \tag{4.19}$$

$$\tau_r \frac{dr_{lx}}{dt} = (1-\beta)\varepsilon_{lx} - \beta\delta_{lx}, \tag{4.20}$$

$$\tau_r \frac{dr_{ly}}{dt} = (1-\beta)\varepsilon_{ly} - \beta\delta_{ly}, \tag{4.21}$$

$$\tau_\delta \frac{d\delta_{lx}}{dt} = r_{lx} - r_{u\rho}\cos r_{u\theta} - \delta_{lx}, \tag{4.22}$$

$$\tau_\delta \frac{d\delta_{ly}}{dt} = r_{ly} - r_{u\rho}\sin r_{u\theta} - \delta_{ly}, \tag{4.23}$$

$$\tau_\varepsilon \frac{d\varepsilon_{u\rho}}{dt} = \sqrt{{r_{lx}}^2 + {r_{ly}}^2} - r_{u\rho} - \varepsilon_{u\rho}, \tag{4.24}$$

$$\tau_\varepsilon \frac{d\varepsilon_{u\theta}}{dt} = \arctan\frac{r_{ly}}{r_{lx}} - r_{u\theta} - \varepsilon_{u\theta}, \tag{4.25}$$

$$\tau_r \frac{dr_{u\rho}}{dt} = (1-\beta)\varepsilon_{u\rho} - \beta\delta_{u\rho}, \tag{4.26}$$

$$\tau_r \frac{dr_{u\theta}}{dt} = (1-\beta)\varepsilon_{u\theta} - \beta\delta_{u\theta}, \tag{4.27}$$

$$\tau_\delta \frac{d\delta_{u\rho}}{dt} = r_{u\rho} - V_\rho - \delta_{u\rho}, \tag{4.28}$$

$$\tau_\delta \frac{d\delta_{u\theta}}{dt} = r_{u\theta} - V_\theta - \delta_{u\theta}, \tag{4.29}$$

where A_x and A_y represent the Cartesian input while V_ρ and V_θ represent the polar top-down prediction.

Let $\vec{S}$ represent state variables in the system where

$$\vec{S} = [\varepsilon_{lx}, \varepsilon_{ly}, r_{lx}, r_{ly}, \delta_{lx}, \delta_{ly}, \varepsilon_{u\rho}, \varepsilon_{u\theta}, r_{u\rho}, r_{u\theta}, \delta_{u\rho}, \delta_{u\theta}]^T. \tag{4.30}$$

A stability analysis can be produced by finding the eigenvalues of the system matrix for the linear approximation about an equilibrium. Using the same derivation in Section 3.3 to find the stability, let $\vec{S}^*$ be an equilibrium point. Let $F(\vec{S})$ represent the system of dynamic equations, where the linear approximation is

$$F(\vec{S}) \approx \nabla F(\vec{S}^*)\Delta\vec{S}. \tag{4.31}$$

Thus, the dynamics of the SPE system can be approximated by

$$\tau\frac{d\Delta\vec{S}}{dt}=\begin{bmatrix}\Delta\dot{\varepsilon}_{lx}\\ \Delta\dot{\varepsilon}_{ly}\\ \Delta\dot{r}_{lx}\\ \Delta\dot{r}_{ly}\\ \Delta\dot{\delta}_{lx}\\ \Delta\dot{\delta}_{ly}\\ \Delta\dot{\varepsilon}_{u\rho}\\ \Delta\dot{\varepsilon}_{u\theta}\\ \Delta\dot{r}_{u\rho}\\ \Delta\dot{r}_{u\theta}\\ \Delta\dot{\delta}_{u\rho}\\ \Delta\dot{\delta}_{u\theta}\end{bmatrix}=\begin{bmatrix}-1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 1-\beta & 0 & 0 & 0 & -\beta & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1-\beta & 0 & 0 & 0 & -\beta & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & -\cos r_{u\theta} & r_{u\rho}\sin r_{u\theta} & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -\sin r_{u\theta} & -r_{u\rho}\cos r_{u\theta} & 0 & 0\\ 0 & 0 & \frac{r_{lx}}{\sqrt{{r_{lx}}^2+{r_{ly}}^2}} & \frac{r_{ly}}{\sqrt{{r_{lx}}^2+{r_{ly}}^2}} & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0\\ 0 & 0 & \frac{r_{ly}}{\sqrt{r_{lx}^2+r_{ly}^2}} & \frac{r_{lx}}{\sqrt{r_{lx}^2+r_{ly}^2}} & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 1-\beta & 0 & 0 & 0 & -\beta & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1-\beta & 0 & 0 & 0 & -\beta\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1\end{bmatrix}\begin{bmatrix}\Delta\varepsilon_{lx}\\ \Delta\varepsilon_{ly}\\ \Delta r_{lx}\\ \Delta r_{ly}\\ \Delta\delta_{lx}\\ \Delta\delta_{ly}\\ \Delta\varepsilon_{u\rho}\\ \Delta\varepsilon_{u\theta}\\ \Delta r_{u\rho}\\ \Delta r_{u\theta}\\ \Delta\delta_{u\rho}\\ \Delta\delta_{u\theta}\end{bmatrix}. \tag{4.32}$$

Suppose we set $\beta=\frac{1}{2}$, and choose input $(V_\rho,\, V_\theta) = \left(\sqrt{A_x^2+A_y^2},\, \arctan\frac{A_y}{A_x}\right)$ and $(A_x,\, A_y)$ $= (V_\rho\cos V_\theta,\, V_\rho \sin V_\theta)$. It can be shown that $\vec{S_1}$ is an equilibrium solution where

$$\begin{aligned}\vec{S_1} &= [\varepsilon_{lx}, \varepsilon_{ly}, r_{lx}\qquad, r_{ly}\qquad, \delta_{lx}, \delta_{ly}, \varepsilon_{u\rho}, \varepsilon_{u\theta}, r_{u\rho}\qquad, r_{u\theta}\qquad, \delta_{u\rho}, \delta_{u\theta}]^T, &(4.33)\\ &= [0\ \ , 0\ \ , V_\rho\cos V_\theta, V_\rho\sin V_\theta, 0\ \ , 0\ \ , 0\ \ , 0\ \ , \sqrt{A_x^2+A_y^2}, \arctan\frac{A_y}{A_x}, 0\ \ , 0\ \ ]^T. &(4.34)\end{aligned}$$

At that equilibrium solution, the eigenvalues of the system matrix (4.32) are,

$$\begin{bmatrix} -\frac{1}{2}-\frac{1}{2}i \\ -\frac{1}{2}+\frac{1}{2}i \\ -\frac{1}{2}-\frac{1}{2}\sqrt{5}i \\ -\frac{1}{2}+\frac{1}{2}\sqrt{5}i \\ -\frac{1}{2}-\frac{1}{2}i \\ -\frac{1}{2}+\frac{1}{2}i \\ -\frac{1}{2}-\frac{1}{2}\sqrt{5}i \\ -\frac{1}{2}+\frac{1}{2}\sqrt{5}i \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}. \qquad (4.35)$$

The fact that the real part of all the eigenvalues is negative, indicate the system would be pushed toward the equilibrium given a perturbation from the steady-state solution [28]. Hence, the SPE network is stable for a network that models a Cartesian-to-polar transform and *vice versa*. Contrast this to the PE model in Section 3.3 where only a certain range of θ values are stable.

Chapter 5

Implementation

In this chapter, we discuss the software implementation for the analytical and NEF model for the SPE. The analytical model was used to simulate the system of differential equations, while the NEF model allows for the implementation into a functional brain model using spiking neurons. Both implementations were run on a OS X machine with a 2.4Ghz Intel Core i7 processor and 8 GB of RAM.

5.1 Analytical Model

The SPE unit was built as a Python module that encompasses ε, r and δ as state variables, and f and g as functional mappings. Experiments were conducted by simulating small time steps, dt, that would translate into incremental updates to the SPE unit variables using the differential equations (4.1)-(4.3). Euler's method was used to approximate the solution of the system of differential equations. Randomly generated samples of sensory input were provided to the lower level SPE units as bottom-up input, while the corresponding top-down predictions were provided to the top-level SPE units as top-down input. A sample was held for a constant amount of time steps before switching to another sample.

A simulation consists of initializing SPE modules, and linking the SPE units together with references to neighboring f or g output functions which are utilized to calculate updates to state variables in (4.2) and (4.3).

5.1.1 Functional Mappings

Recall that the feed-forward and feed-back mappings f and g were defined as a linear combination of coefficients and basis functions in Section 4.2 . In the analytical model, the mappings utilized a "Tent" linear-interpolation-basis function shown in Figure 5.1. The equation for the basis function is defined as

$$\phi_n(r) = \begin{cases} \frac{(r-D_{n-1})}{\Delta D} & D_{n-1} \leq r < D_n, \\ \frac{(D_{n+1}-r)}{\Delta D} & D_n \leq r < D_{n+1}, \\ 0 & otherwise. \end{cases} \tag{5.1}$$

This function provides a linear interpolation based on the nearest discrete domain points

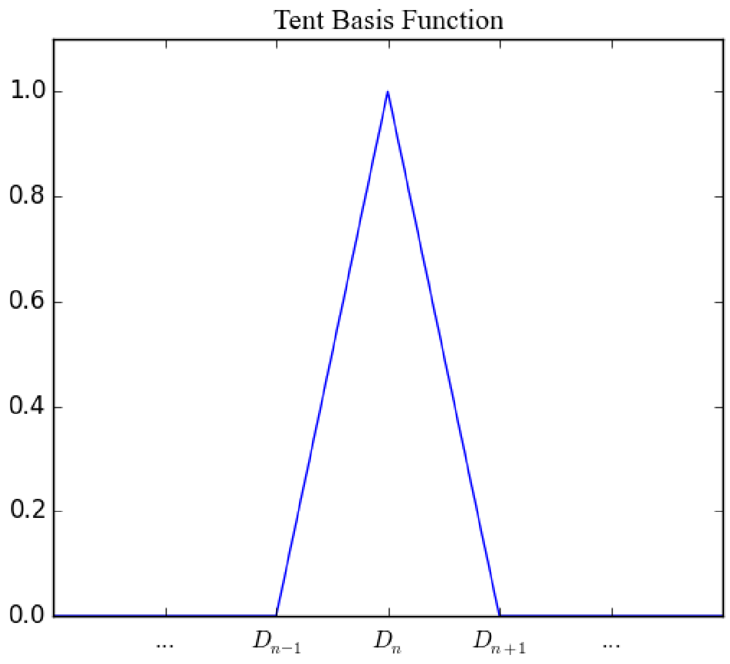


Figure 5.1: A linear combination of N "Tent" basis functions, and coefficients can used to approximate a function.

from D_1 to D_N. The functional mappings f and g are implemented as a sum of the basis functions multiplied by coefficients b_n and c_n respectively. Figure 5.2 shows the b_n weight coefficients for a learned 1-dimensional surface map for the identity function, with size

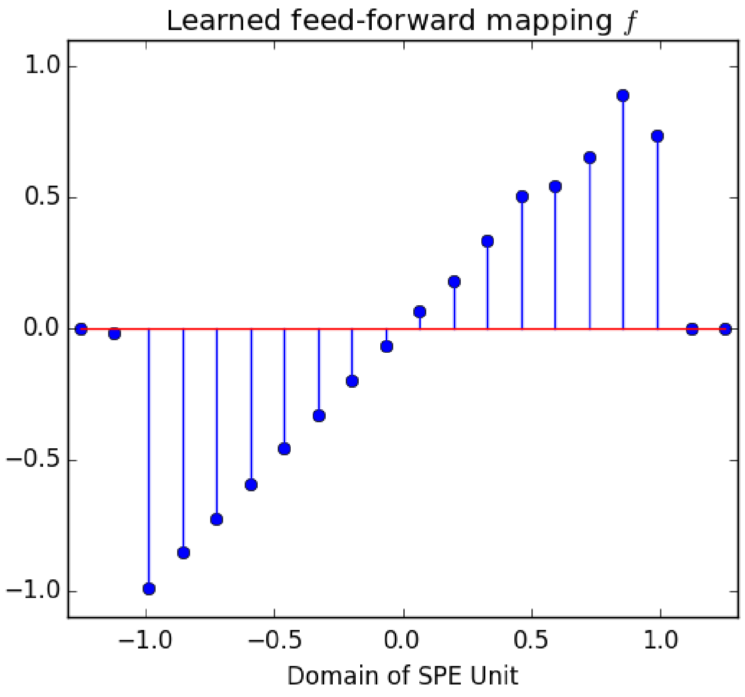


Figure 5.2: Example of feed-forward mapping f as a combination of N=20, "tent" basis functions scaled by b_n. The domain limits are from D_1 = -1.25 to D_{20} = 1.25, where ΔD = $\frac{2.5}{19}$. The mapping was only trained on values ranging from -1 to 1.

N = 20. Because the basis function is a linear interpolation of the two nearest points, evaluating the value of the function can be done using linear interpolation as well. For example, given the learned identity function in Figure 5.2, evaluating f at r=0.23 would result in an interpolation between n=11 and n=12 where $D_{11} \approx 0.183$ and $D_{12} \approx 0.335$. The result is $f(0.23) \approx 0.229$. This allows us to evaluate the f and g functions without evaluating every domain point, resulting in faster simulation times.

5.2 Neural Engineering Framework (NEF) Model

The SPE model was implemented into Nengo version 2.1.0. Nengo is a Python implementation of the NEF, which allows for easy construction and simulation of neural models [3]. The SPE structure could be translated into Nengo with minor adjustments. Refer to "Appendix A" for a explanation of how values and connections are represented in the framework. The state variables ε, r and δ are represented as neural ensembles in Nengo, while the learning rules for f and g change the decoding values using the Prescribed Error Sensitivity learning rule [4]. Every node in the analytical model is implemented as neural ensembles in Nengo. Like the analytical model, a simulation was defined by initializing the SPE units inside of a Nengo simulation, connecting the SPE units with each other, and connecting bottom-up and top-down input nodes to the SPEs. The simulation would run for a given amount of time T, with randomly generated input samples held for a short period of time.

5.2.1 Structure

Similar to the analytical model, the components of the SPE were combined into a Nengo SPE module that encompassed the state variables as neural ensembles. The Nengo SPE is outlined in Figure 5.3. Auxiliary ensembles *post* ε_i and *post* δ_i were required to facilitate scaling the ε_i and δ_i ensembles by β. In Nengo, the learning rules are turned on and off by inhibiting the activity of the neurons that supply the error signal. Inhibiting the activity of ε_i or δ_i would subsequently change the state values of ε_i and δ_i to zero within the SPE system, which is not a desired effect. This is circumvented by creating auxilary ensembles ε_i *copy* and δ_i *copy* as the error nodes used for learning. The synaptic time constant for the auxiliary ensembles were set to 0ms in order to remove delay caused by the additional synapse. These auxiliary ensembles are for testing purposes and are not inherent to the structure of the SPE.

5.2.2 NEF Learning Rule

The Prescribed Error Sensitivity (PES) learning rule is an error-based learning rule that is implemented in Nengo [4], that is derived using gradient descent [26]. Adjustments are made to decoding weights based on an error value. This fits the SPE model nicely as there are two readily available nodes, ε and δ, that provide the error value.

The general PES learning rule in Nengo is defined in Bekolay et al. [4] as

$$\Delta w_{ij} = \kappa \alpha_j e_j \cdot E a_i, \tag{5.2}$$

where w_{ij} represent connection weight from neuron i to neuron j, α_j is a scaling factor, e_j is the NEF encoding vector, E is the error term and a_i is the activity of the pre-synaptic neuron.

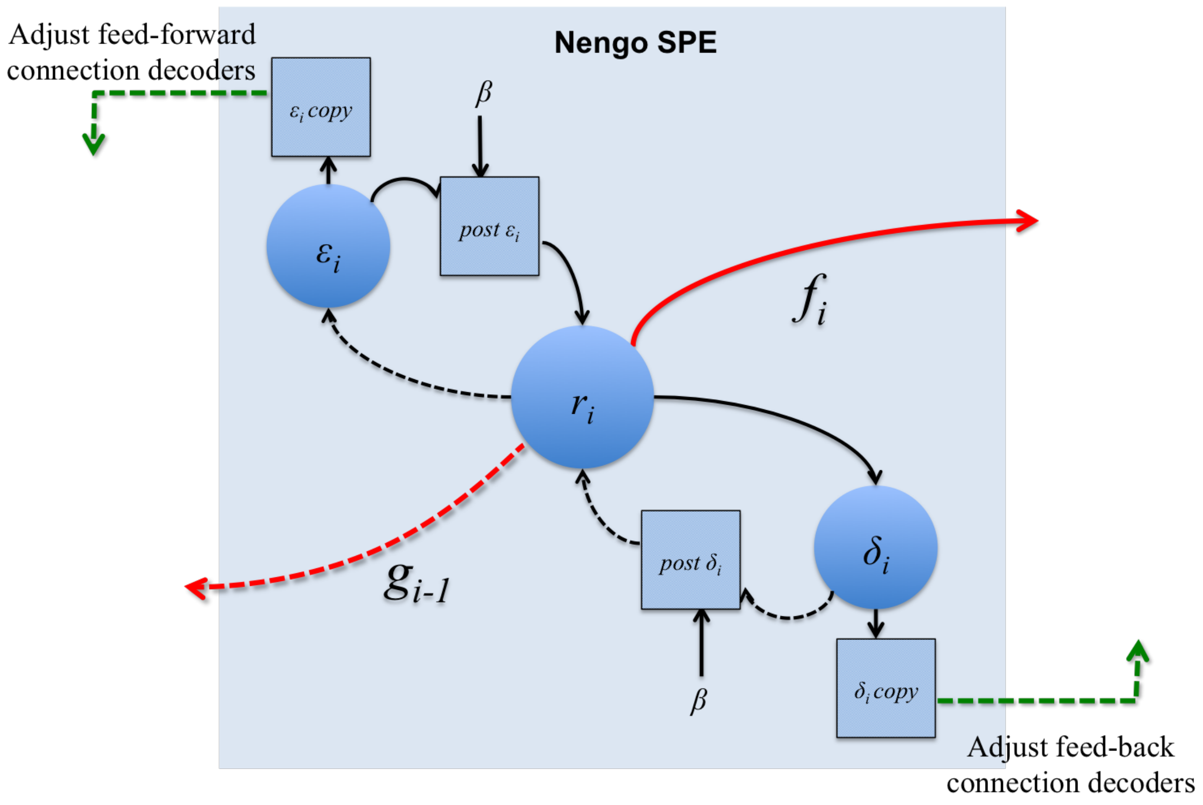


Figure 5.3: SPE Unit implemented into Nengo. Each node is implemented as a neural ensemble in Nengo, where a population of neurons represent the value. Square nodes with a black border represent auxiliary ensembles that were added to Nengo in addition to the analytical SPE model. These ensembles are for testing purposes and are not inherent to the SPE model.

This learning rule shares similar traits to the generalized learning rule for the SPE presented in Section 4.3 as (4.14). The change in weights w_{ij} bear similar meaning to the weight coefficients of the basis function, $\vec{b}$ and $\vec{c}$. The error term E is proportionate to error nodes ε_{i+1} and δ_i. Changes in the weights for the PES learning rule stem from changes in the decoding weights multiplied by the encoders. The decoding weights in the NEF are essentially the same as the weight coefficients of the basis functions.

Chapter 6

Results and Evaluation

6.1 SPE Analytical Model

The analytical SPE model allows us to quickly prototype and explore the dynamics of an SPE network. The following experiments investigate the ability for the SPE to learn the b_n and c_n coefficients for the f and g function respectively, such that the activity of the error nodes ε and δ are minimized. In addition we explore how the β parameter affects the network. At the beginning of each simulation, the b_n and c_n coefficients are initiated with random values. While the simulation runs, the coefficient values change based on the dynamics equations for learning as outlined in (4.14) and (4.15). For the following experiments in this section, a time-step of dt=0.001ms and a time constant τ=0.05s was used.

6.1.1 Identity Mapping

A basic test for the SPE units is to see if a 1-dimensional identity mapping can be learned, and if the network can remain stable. This experiment consists of two SPE units, l and u, which receive identical input of values ranging from -1 to 1. A network diagram of this setup is shown in Figure 6.1. The network was trained for a total time of T=300s with a β=0.5. Samples were held for 5 seconds each, a learning rate of K=10 was used, and N=20 coefficients were used for both f and g. The domains $\{D_1, D_{20}\}$ for the f and g transform were set to [-1.2, 1.2].

Table 6.1 shows examples of the input A and V that are held for 5 seconds. Figure

Table 6.1: Example of input values for A and V for the SPE units to learn the identity mapping.

Sim. Time	Input (A)	Input (V)
0.0	0.45	0.45
5.0	-0.11	-0.11
10.0	0.73	0.73
...	...	...

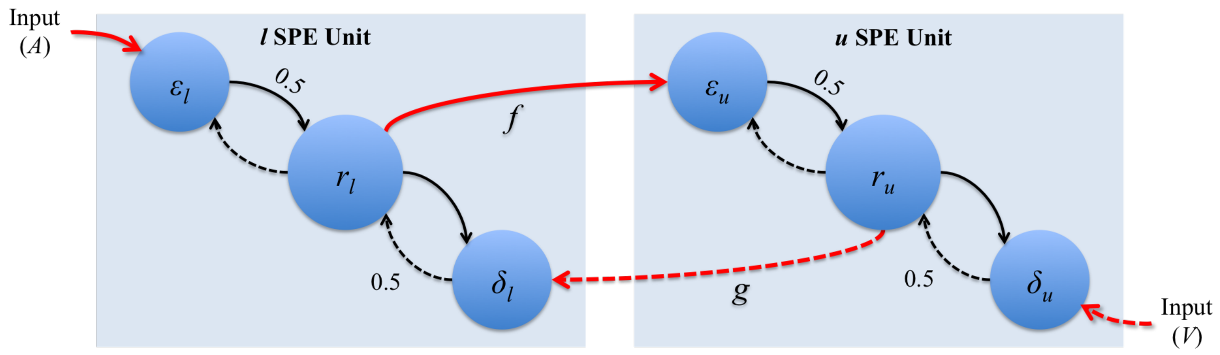


Figure 6.1: Network of SPE units set up to learn the identity mapping. The weighting parameter β is set to 0.5, which weights the feed-forward and feed-back mappings equally at 0.5. The A and V inputs represent the bottom-up input and top-down expectation respectively, where A=V. The identity transformation is the ideal function in which the f and g functions should learn in order to minimize residual errors.

6.2 shows the first 50 seconds of the simulation (50000 time-steps) and the state representations r_l and r_u compared to the input A and V. Observe that the values of each state representation do not match the input at first, but over time the learning rules drive the f and g mappings to minimize error. Figure 6.3 shows the progression of the b coefficients for the f mapping. As the simulation progresses the feed-forward mapping f starts to resemble the identity function $f(x) = x$. The same was observed for the g mapping shown in Figure 6.4.

Figure 6.5 shows the error units used in calculating the coefficients for b_n and c_n. They are being driven to zero over time because the f and g mappings are beginning to match the identity function. We can demonstrate that the network is functional with the learned mappings. In order to do this, the learning is turned off by not applying the updates to the

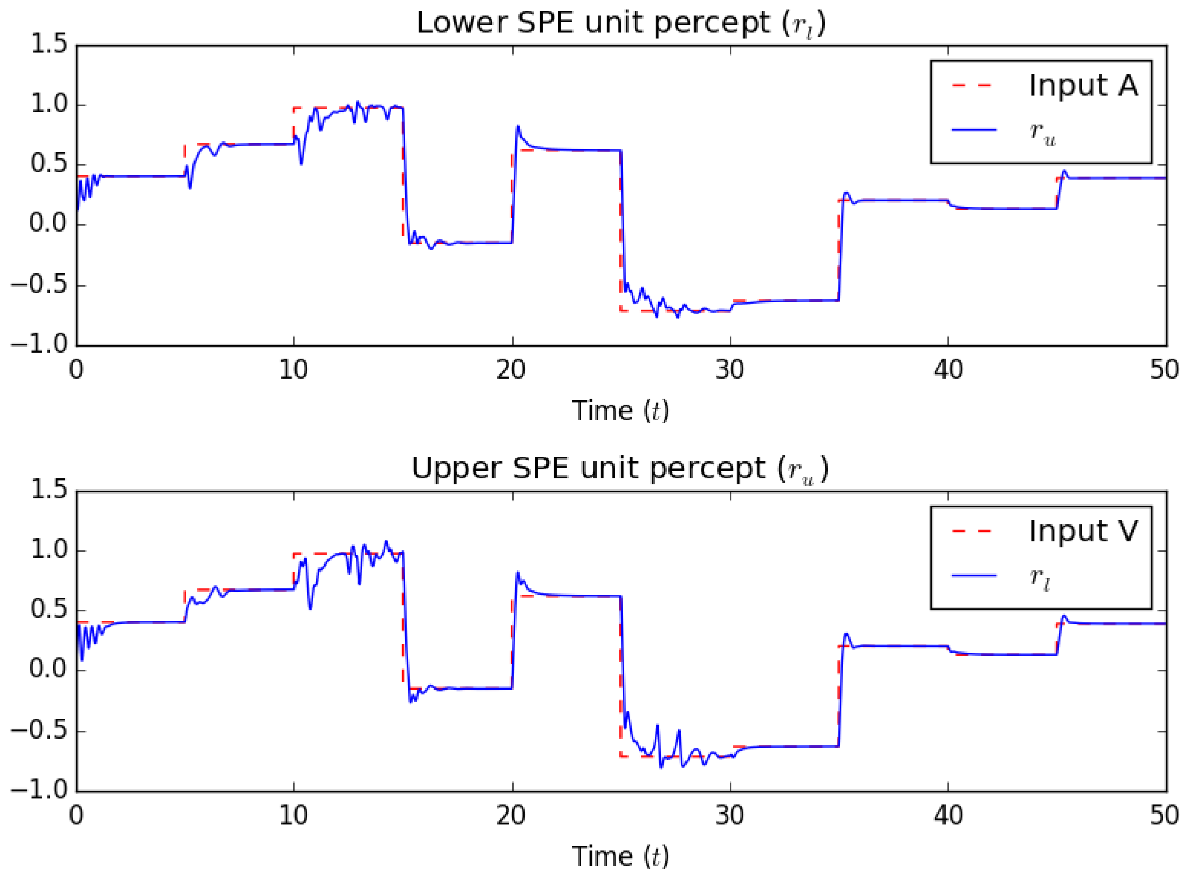


Figure 6.2: The values of each state representation start to match their respective inputs over time for each new sample.

dynamical system made by (4.14) and (4.15). We can also drive the network in a purely feed-forward fashion by setting $\beta = 0$. Conversely the network can be purely feed-back by setting $\beta = 1$. This allows us to test the dynamics of the network and state representations as if there were no bottom-up input A, or top-down input V. Figure 6.6 shows that the system can be driven in a purely feed-forward or feed-back manner and still maintain the the correct state representation values in both the lower and upper layer.

The identity mapping experiment showed the basic functions of the SPE unit. It has the ability to perform learning in a continuous manner of changing inputs, and at the same time learn individual f and g mappings. It is important to note that the g mapping is

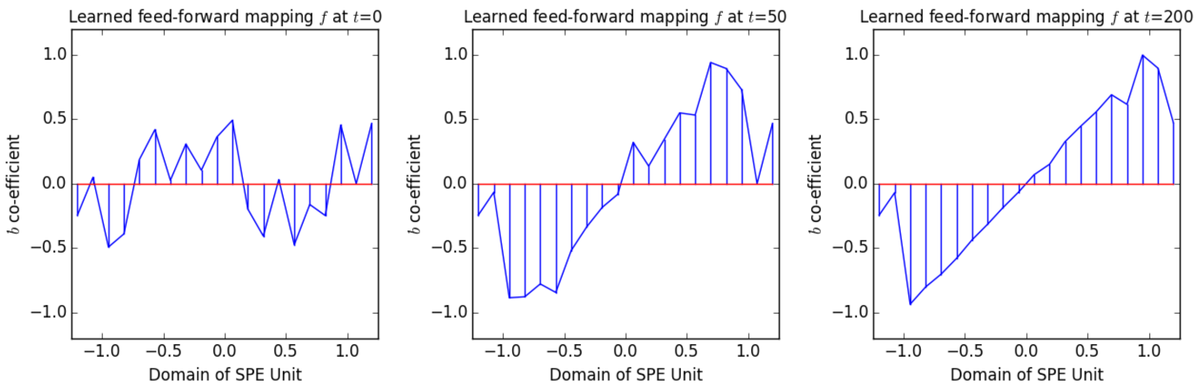


Figure 6.3: The b coefficents are random at first (left), and as the simulation runs they are gradually adjusted to minimize error ε_u (middle). By the end of the simulation the coefficients start to match the identity function for a range from -1 to 1 (right).

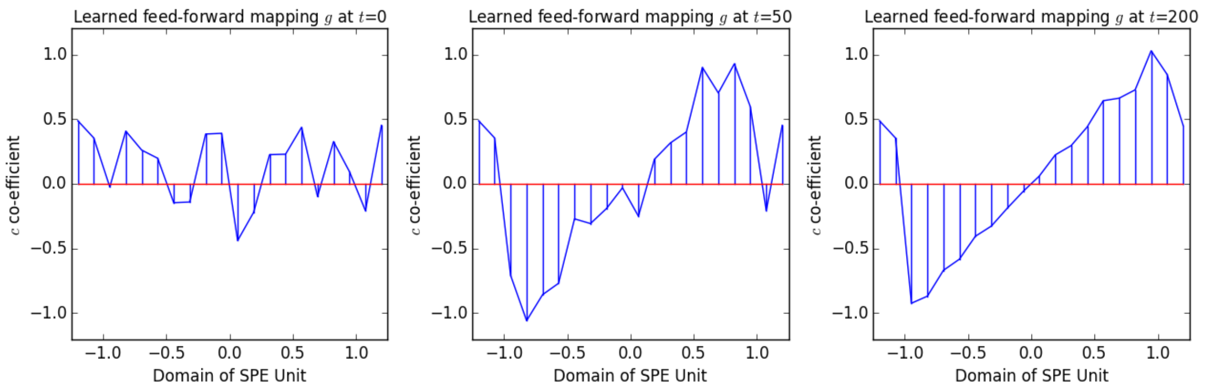


Figure 6.4: The c coefficents are random at first (left), and as the simulation runs they are gradually adjusted to minimize error δ_l (middle). By the end of the simulation the coefficients start to match the identity function for a range from -1 to 1 (right).

not copying the weights of the f mapping and vice-versa. By modulating the β value, we also show the versatility of a network of SPE units. The correct value of each state representation can still be generated if one form of input is cut-off from the network, which is shown in Figure 6.6.

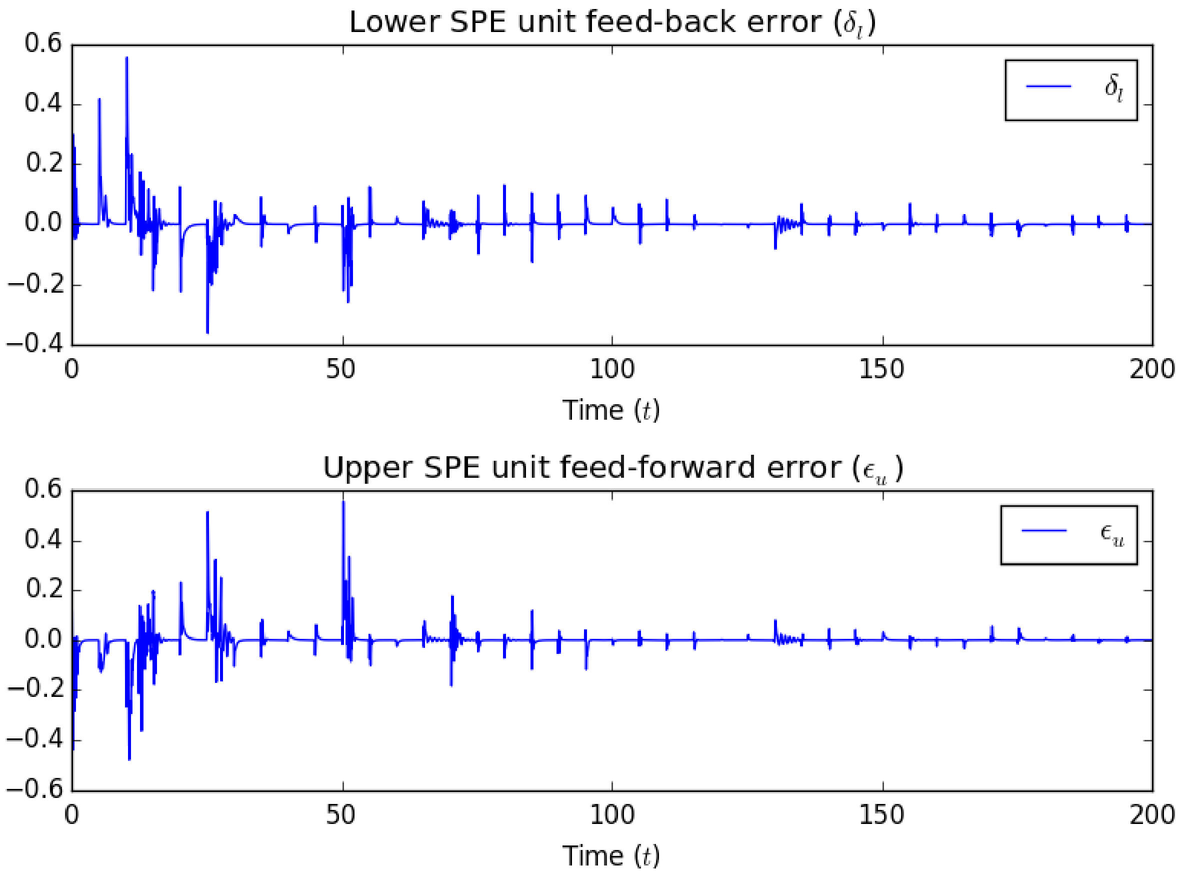


Figure 6.5: The error nodes δ_l and ε_u get closer to zero with each new sample input. This is because f and g are changing in order to minimize the errors.

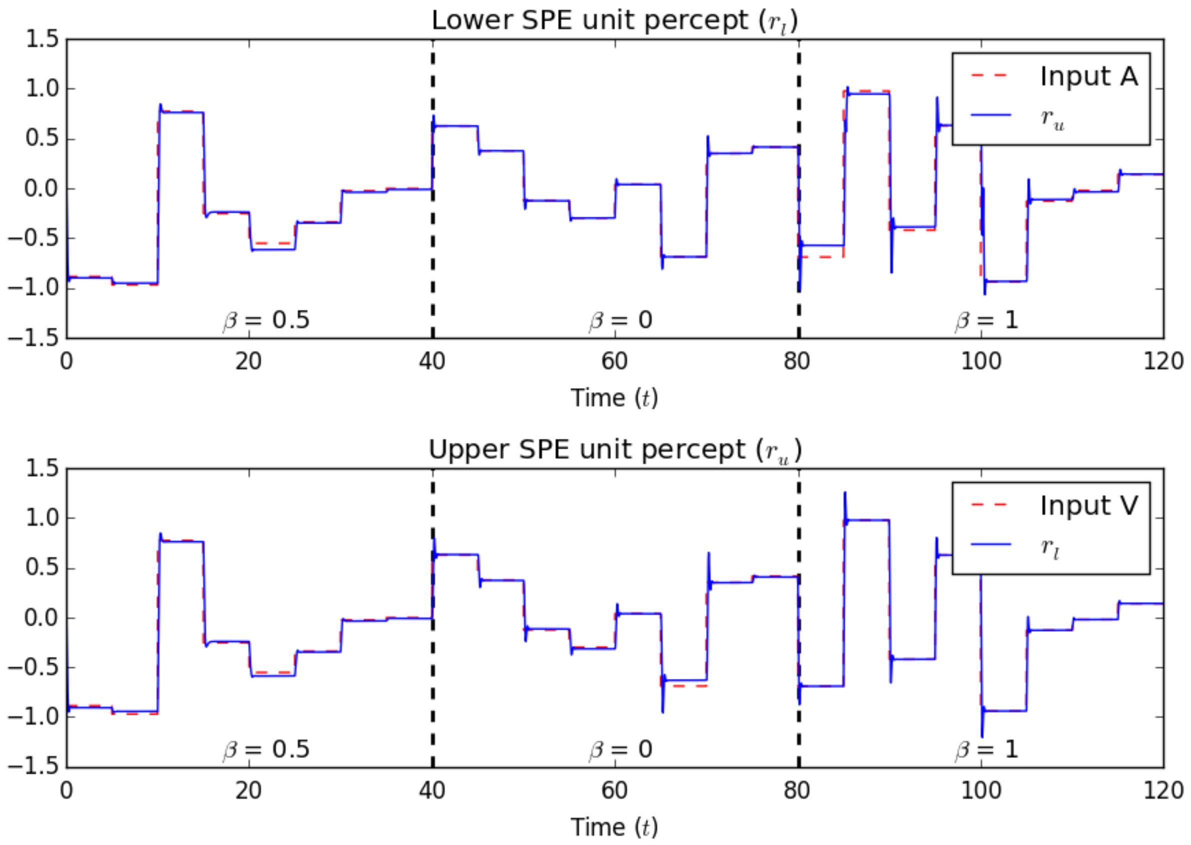


Figure 6.6: When the simulation is run with learning turned on, the network makes weight adjustments to learn f and g functions. We can then run the simulation with the learning turned off and we can show how the network reacts with different β values. The first 40 seconds runs with $\beta = 0.5$ (balanced weighting). The state representations of both the lower and upper SPE do a good job at matching the inputs. From 40s to 80s, $\beta = 0$ sets the network into a purely feed-forward mode. During this time the lower SPE matches the input perfectly, while the upper SPE is slightly off due to the non-ideal f function. The same can be said about the reverse non-ideal g function. From 80s to 120s, $\beta = 1$ and the lower SPE is slightly off, while the upper SPE matches the input.

6.1.2 Cartesian and Polar Mapping

A focal point of the SPE model is its ability to learn individual non-linear feed-forward and feed-back transformations simultaneously, and have the network remain stable. As discussed in Section 4.4, an example scenario is the Cartesian-to-polar transformation. The experimental model is shown in Figure 4.2. The network was trained with T=3000s with a β=0.5. Samples were held for 2.5 seconds each, and a learning rate of K=50, and N=50 coefficients were used for both f and g. The learning rate K and number of coefficients n were increased relative to the identity experiment to facilitate learning a more complex function.

The domains $\{D_1, D_{50}\}$ of the f transform were set to [-1, 1] for both components, while the domains of the g transform were set to [0, 1] and [-π, π] for the two components. Table 6.2 shows examples of input used to train the network. The sample input range for V_ρ was from 0 to 1 and V_θ was from -π to π.

Table 6.2: Example input values for A_x, A_y, V_ρ and V_θ for the SPE units to learn the Cartesian and polar mapping.

Sim. Time	Input (A_x)	Input (A_y)	Input (V_ρ)	Input (V_θ) in rad
0.0	0.478	-0.043	0.480	-0.090
2.5	0.101	-0.164	0.193	-1.022
5.0	-0.430	-0.489	0.651	-2.293
...	...	...	...	...

Figure 6.7 and Figure 6.8 show the growth of the f and g mappings over the course of the simulation. They begin to resemble the ideal f and g mappings shown in Figure 6.9 and Figure 6.10.

Aside from just a visual inspection of the mapping functions, we are able to quantitatively measure how accurate the learned mappings are. One method to measure accuracy is by calculating the Root Mean Squared Error (RMSE). However, we are simply looking to see if the error decreases over time, and we are not making comparisons to other RMSE values. By testing the learned f and g function with sample test inputs across the domain, we can calculate the RMSE as

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(z_i - \hat{z}_i)^2}, \tag{6.1}$$

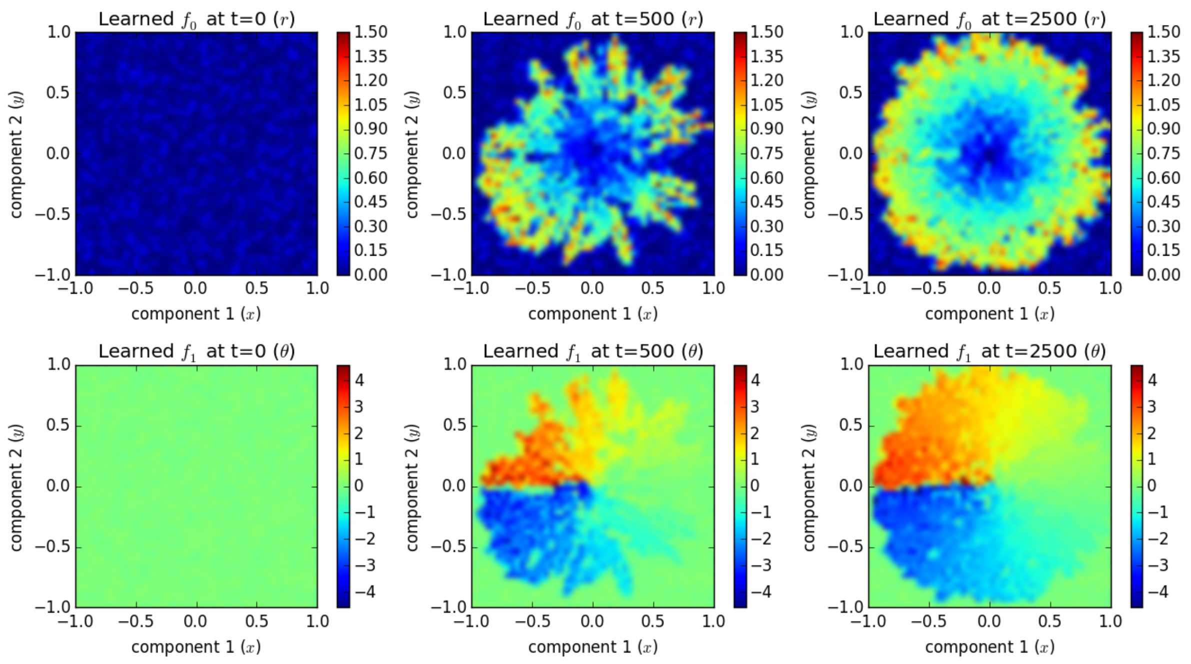


Figure 6.7: The f function is a 2-D mapping of two components f_0 (top) and f_1 (bottom). The left plots show the b coefficients which are initiated randomly at t=0. At t=500s, the coefficients start to resemble the Cartesian-to-polar transformation (middle), and at t=2500 the transformations are very close the ideal transformation (right).

where m represents the number of test inputs, and i denotes the index of the test input. When evaluating the g function, z_i represents the Cartesian co-ordinate from the ideal transform, and $\hat{z}_i$ represents the output from g. When evaluating the f function, z_i represents the input Cartesian co-ordinate, while $\hat{z}_i$ represents the output polar co-ordinate from f converted to a Cartesian co-ordinate using the ideal transform. The error in this experiment is calculated by the Euclidian distance between z_i and $\hat{z}_i$. An amount of m=100 test inputs across the domain were used. Figure 6.11 shows the RMSE calculated in 25s intervals during the simulation. The results show that the RMSE decreases for both f and g and at a rate proportional to each other.

This experiment demonstrated that the SPE model can simultaneously learn 2-dimensional non-linear functions over time. The accuracy of the function mappings begin to increase over time, however, many more sample inputs were required to get close to the ideal mapping compared to the amount of samples used to learn the identity function.

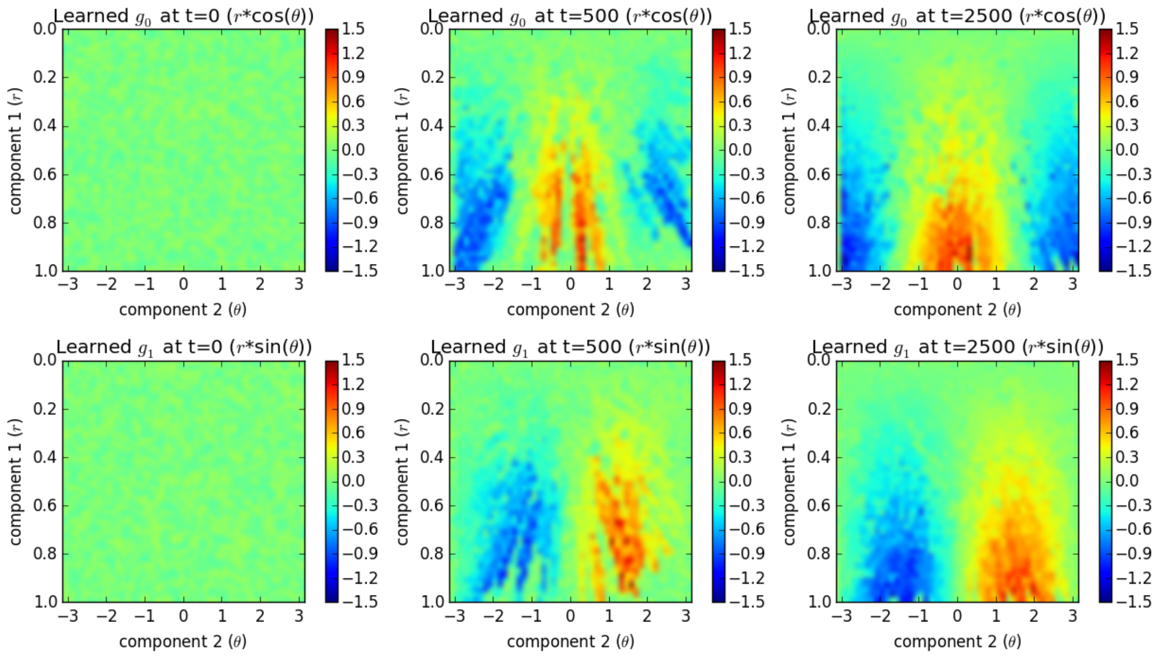


Figure 6.8: The g function is a 2-D mapping of two components g_0 (top) and g_1 (bottom). The left plots show the c coefficients which are initiated randomly at t=0. At t=500s, the coefficients start to resemble the polar-to-Cartesian transformation (middle), and at t=2500 the transformations are very close the ideal transformation (right).

This experiment contrasts the ability for the SPE model to learn the full range of polar to Cartesian mappings, compared to the PE model which could not. These results are expected with what we found in the stability analysis in Section 4.4.

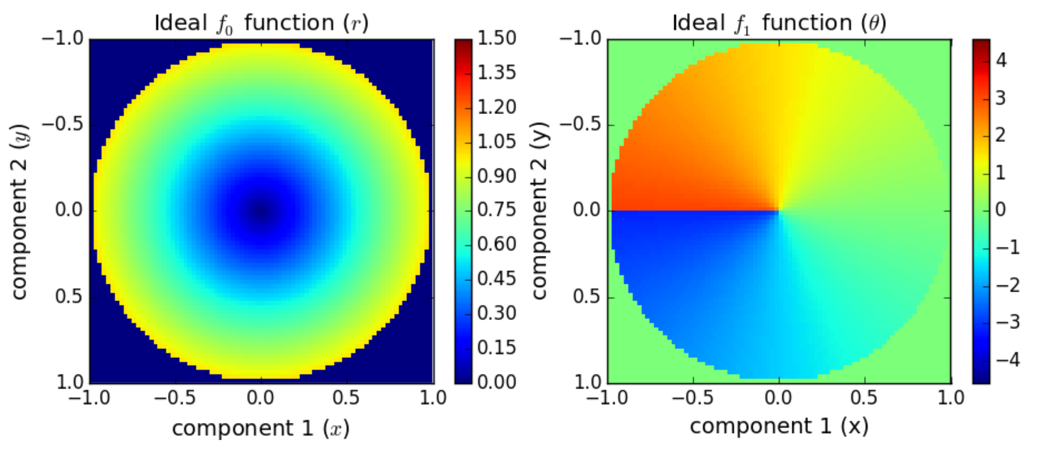


Figure 6.9: The ideal Cartesian-to-polar transformation for the range of input values. The first component (left) is the radius which is calculated from the x and y Cartesian co-ordinates, and the second component (right) is the angle which is calculated using the $arctan$ function.

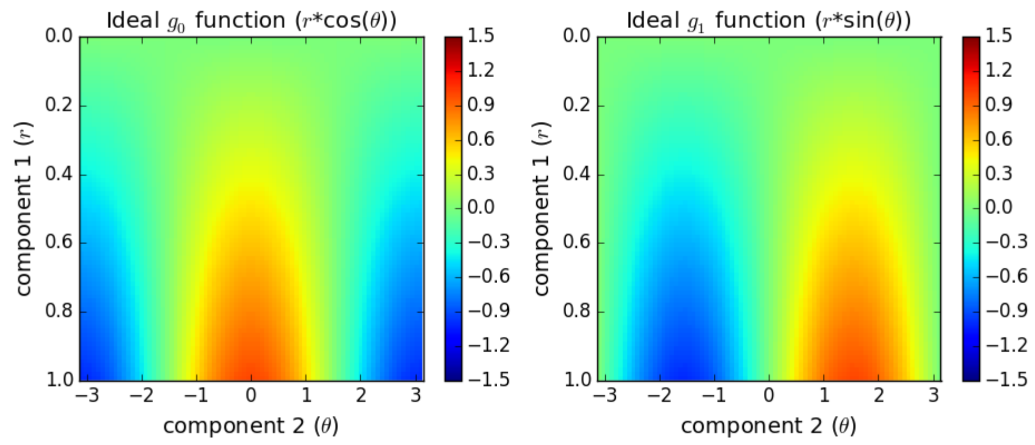
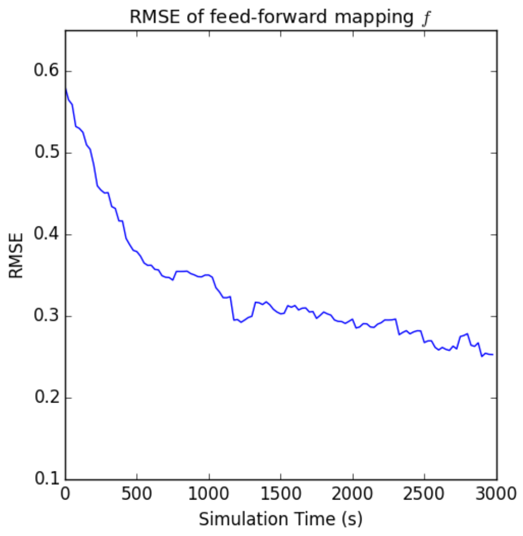


Figure 6.10: The ideal polar-to-Cartesian transformation for the range of input values. The first component (left) is the x co-ordinate which is calculated using $r * \cos\theta$, and the second component (right) is the y co-ordinate which is calculated using the $r * \sin\theta$.



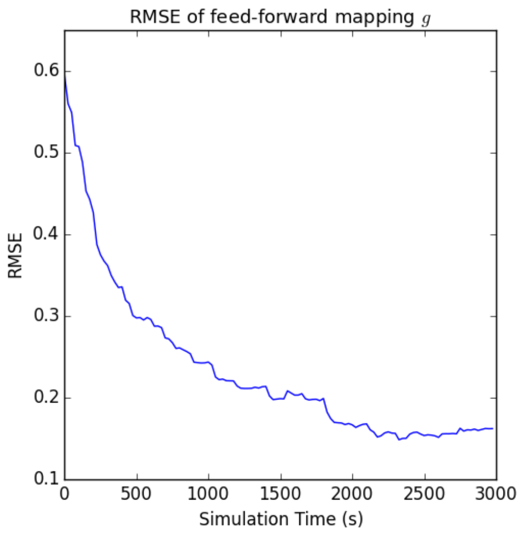


Figure 6.11: The RMSE for sample inputs to the learned f (left) and g (right) function over time.

6.1.3 Exclusive OR

How does the SPE network react if the inputs to the system have a non-invertible f function? The common logical operator called the *exclusive or* (XOR) does not have an inverse operation. Table 6.3 summarizes the XOR operation and the experimental setup for the SPE system. The SPE model is similar to the Cartesian-to-polar experiment, except the upper level SPE is 1-dimensional instead of 2-dimensional. The bottom-up input A_X and A_Y are randomly generated Cartesian points centered at their respective values from Table 6.3, while the top-down input $V_{X \oplus Y}$ is either 1 or -1 depending on the quadrant the input is in.

It is expected that the lower level state representation r_{lX} and r_{lY} will hold the bottom-up A_X and A_Y values, and the upper level state representation r_u will hold the XOR value. However, it is clear that no ideal function g exists such that the value of r_u can be transformed into the correct r_{lX} and r_{lY} values.

This experiment demonstrates the SPE's ability to learn a feed-forward mapping when an inverse does not exist. This can also be thought of as a method of supervised machine learning. Because there is no ideal g mapping, the lower SPE will have a non-zero δ_l which will consistently change the feed-back coefficients c_n. Since it is the feed-forward function f that will learn the XOR operation, the experiment parameters are tuned such that the network focuses more on the feed-forward mapping. To do this, we set $\beta = 0.1$, leaving a 0.9 weighting on the forward mapping, and 0.1 weighting on the feed-back mapping.

The network was trained for T=3000s, samples were held for 2.5 seconds each with a learning rate of K=5, The SPE's used N=25 coefficients for both f and g, with domains from [-0.5,0.5] for both components of the f function, and [-1.1, 1.1] for the g function. Figure 6.12 shows the learned f function at the end of the simulation. Note that the result represents the desired transform set in Table 6.3. The network was able to learn a

Table 6.3: The truth table for XOR operator, and a table showing the input for both the bottom-up and top-down inputs for the SPE experiment.

XOR Truth Table			SPE Input		
X	Y	$X \oplus Y$	A_X	A_Y	$V_{X \oplus Y}$
0	0	0	0.25	0.25	-1
1	0	1	-0.25	0.25	1
0	1	1	0.25	-0.25	1
1	1	0	-0.25	-0.25	-1

*$\pm$ 0 to .1 noise added to values A_X and A_Y.

representation for f even while the g function could not represent an inverse. We can turn learning off and set $\beta = 0$ and drive the network in a purely feed-forward manner. The resulting simulation yields upper level state representation values close to the actual XOR operation. Figure 6.13 shows the r_u state given some sample inputs when $\beta = 0$.

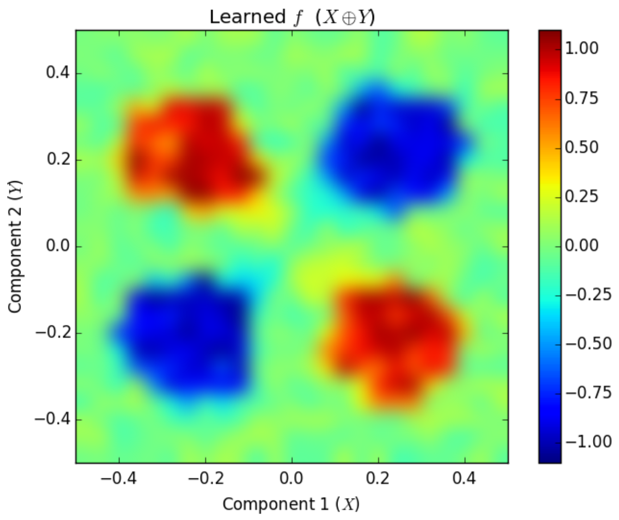


Figure 6.12: The learned feed-forward mapping f when the SPE is trained using inputs resembling the XOR operator.

This experiment demonstrated that the SPE model can perform some supervised-learning tasks. We are able to tune network parameters in such a way to facilitate one-directional learning. Although the error node δ_l is never zero, thus constantly modifying the g function, the SPE network can still generate an accurate feed-forward mapping. This shows the SPE model can perform a simple supervised-learning task, similar to how backpropagation can, but with the use of feed-back connections.

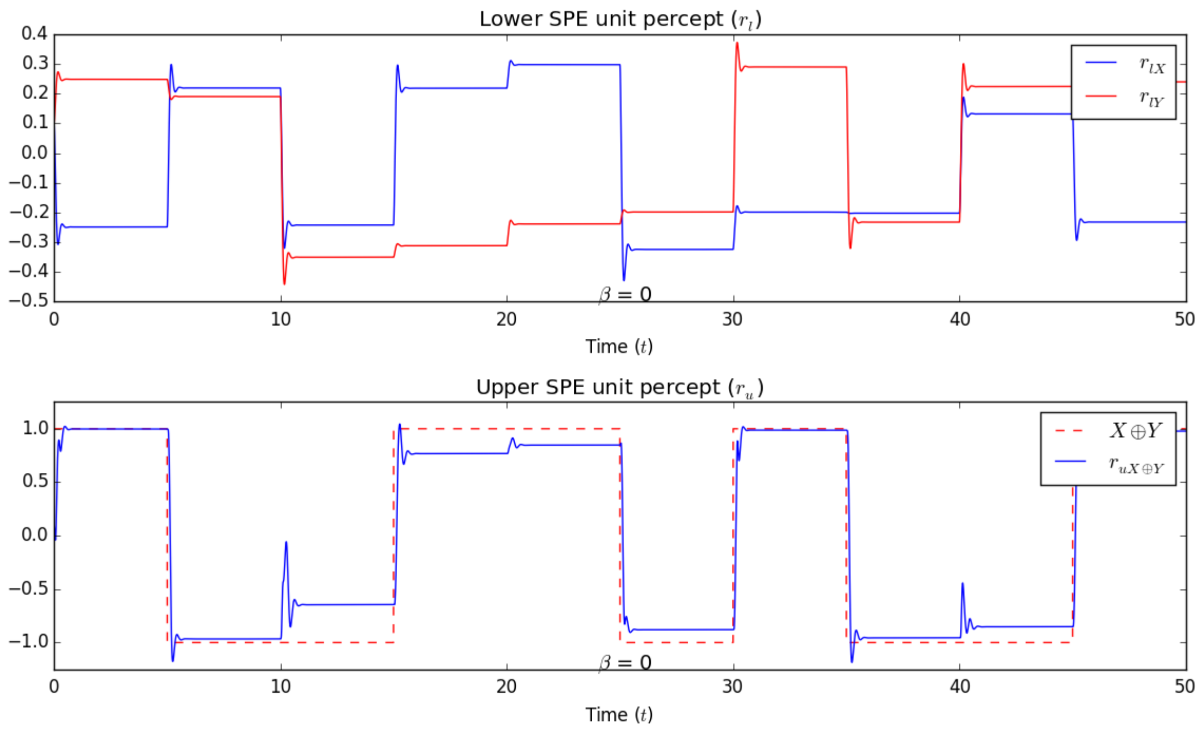


Figure 6.13: The learned feed-forward mapping can be tested by turning learning off, setting β=0, and driving the network with some sample inputs. The lower SPE unit is given sample inputs in each Cartesian quadrant (top). The corresponding response is shown in the upper SPE unit (bottom). Note that like the XOR function, when both r_{lX} and r_{lY} are the same value (or close to), the output is -1, and when they are opposite, the output is 1.

6.2 SPE Nengo Model

The analytical model can be easily translated into a spiking neural model by using the NEF and Nengo. We can run similar experiments from the analytical model in Nengo, and see how the SPE model performs using a biologically plausible framework. The Nengo SPE unit shown in Figure 5.3 was configured with 3000 leaky-integrate-fire neurons per ensemble. The post-synaptic time constants for each connection was set to 0.05 seconds.

6.2.1 Evaluating Learned Mappings

Evaluating the learned connections in Nengo is not as simple as displaying the b_n and c_n coefficients, as it was done in the analytical model. The use of a “Tent” basis function allowed for local changes, and in turn, allowed the use of interpolation between neighboring coefficient values in order to determine the value. On the other hand, the NEF uses biologically inspired neural tuning curves instead. Changes to connection weights affect the full range of representation, rather than just neighboring points. In order to evaluate the learned connections, a uniform sample of data points are sent through the learned connection as input, and the post connection ensemble is probed for the output response. The corresponding input and response is plotted as a binned plot. An example of the output is shown in Figure 6.14.

6.2.2 Cartesian and Polar Mapping

Ideally we would like to reproduce the results we saw in the analytical model using Nengo. Two 2-Dimensional SPE units were set up in a Nengo simulation in a similar fashion as Figure 4.1, but using Nengo SPE units instead. The radius of the lower SPE unit ensembles were set to 1, while the radius of the upper SPE unit ensembles were set to the value of π. The radius refers to the representational space of the ensemble [3].

The learning rate for the PES learning rule was set to 0.0001 and the initial decoding weights for the f and g connection were set randomly. Multiple simulations were run in series, and after each simulation, the learned decoding weights were saved. At the beginning of each simulation, the decoding weights were loaded into the f and g connections. To ensure that these decoding weights would work with each subsequent simulation, the random seed used configure unique properties of each ensemble was carried over, and loaded for the next simulation in the series. In between simulations, the learned f and g functions would be probed with a mesh of data points with the method described in subsection 6.2.1.

Figure 6.14 and 6.15 show the f and g mapping after a total simulation time of 4000 seconds. This is similar to the mappings shown in the analytical model, and is close to the ideal mappings in Figure 6.9 and Figure 6.10. The Nengo mappings look smoother compared to the analytical mappings, because the output values for the Nengo mappings are binned (averaged) responses for the input values. In addition, updates from the learning rule affect the whole range of values instead of just point changes. This experiment showed that the SPE model can be implemented using a spiking neural model and retain similar results to the analytical model.

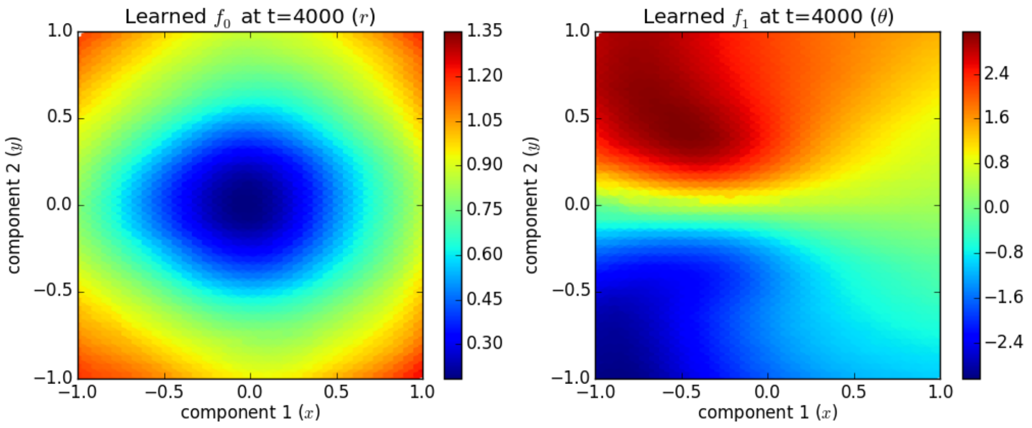


Figure 6.14: The learned connection in Nengo that represents the f transformation after a T=4000 second simulation. The points represent the median values of the post-synaptic output in a 50 by 50 hexagonal bin.

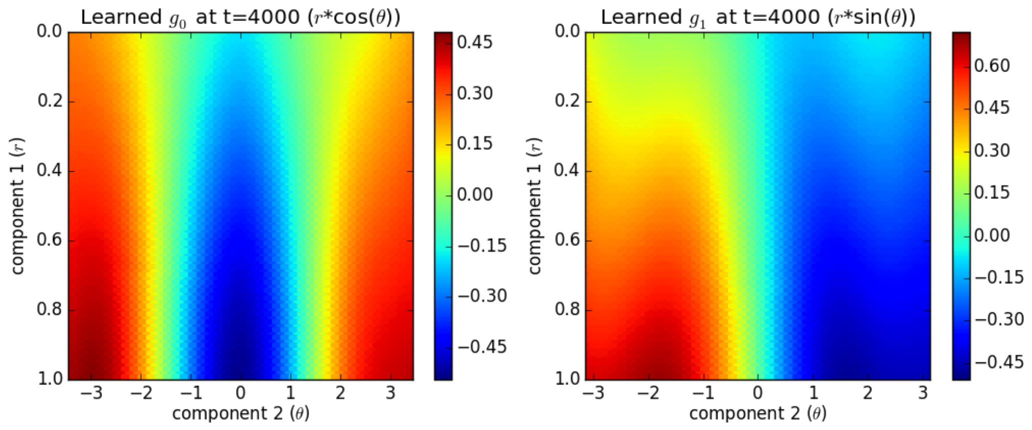


Figure 6.15: The learned connection in Nengo that represents the g transformation after a T=4000 second simulation.

6.2.3 Exclusive OR

The XOR setup from subsection 6.1.3 was replicated into Nengo. The inputs to the SPE units were replicated to match the input conditions outlined in Table 6.3. The learning rate for the PES learning rule was set to 0.001 and the radius of the lower SPE unit was set to 1, and the upper SPE unit to 1.1. The simulation was run for a total of 1500 seconds.

Like the analytical experiment, we also set the β value to 0.1 to emphasize learning on the feed-forward connection. Figure 6.16 shows the learned f mapping.

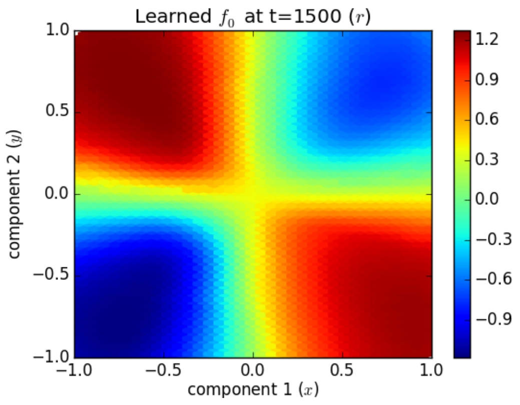


Figure 6.16: The learned connection in Nengo that represents the f transformation after a T=1500 second simulation. The points represent the median values of the post-synaptic output in a 50 by 50 hexagonal bin.

Comparing analytical results in Figure 6.12 to the Nengo model in Figure 6.16, you can see that the f map in Nengo infers information about regions it has not seen. In fact, we can see that the neurons are tuned in a way to actually infer the correct XOR value on points beyond the given inputs. The network was only trained on points at most 0.35 points away from the origin, however the learning rules inferred the correct output values beyond those points.

This experiment demonstrates that the XOR SPE model can be implemented with spiking neurons and neural tuning curves. In fact, this showed a distinct advantage of using tuning curves as a basis function instead of a localized "Tent" function, as it can infer information about the data beyond the sampled points.

Chapter 7

Conclusion and Future Work

In this thesis, we explored the dynamics of a generic PE model. We investigated the PE model using individual feed-forward and feed-back weights and discovered it was unstable for non-linear functions, such as the Cartesian-to-polar case. To solve this, we investigated the dynamics of adding an extra error node to the PE unit, to form the SPE unit. We demonstrated its ability to learn individual feed-forward and feed-back connection weights to facilitate non-linear transformations. In addition, we explored the use of a β parameter to weight the strength of feed-forward and feed-back connections which allows for the network to operate in a purely feed-forward or feed-back basis, or with both forward and backward connections simultaneously. We also demonstrated the SPE's ability to perform a supervised-learning task with the XOR experiment.

There is an abundance of challenges for the SPE model. Our future work is devoted to creating deeper networks where the SPE model spans multiple levels of hierarchies. Unfortunately this posed a few problems. With some preliminary tests, we discovered that the SPE model requires some additional nodes in order to facilitate deeper learning. We found an issue when testing the SPE network with a middle layer (3 layers in total). Experiments showed the middle-layer state held a zero value regardless of the inputs. This un-interesting representation is a result of the middle-layer state being under-constrained. It was observed that the learned f and g connections would output zero values into the middle layer. This would generate zero-valued error nodes ε and δ for the middle layer. Thus, leading to a zero-valued representation.

We are currently exploring methods to add variability to the middle layer representation. By destabilizing the feed-forward and feed-back connections, the SPE can be pushed away from a trivial zero-value representation. In particular, the SPE structure can be modified to hold a moving average, and a standard deviation value. When the standard deviation is near-zero over time (over multiple changing samples), it indicates the f and g functions are not meaningful. Using the standard deviation, the feed-forward and feed-back connections can be modulated to form non-zero transformations. The intuition is to destabilize the representation when there is no variation in the representation despite there being changes to the input.

Another challenge we encountered with the multi-layer network is that the coefficients of the f and g values would sometimes reach increasingly large values in the analytical model. We attempted to address this issue by adding smoothing and zero-mean requirements to the coefficients. Some success was found with basic identity function learning; however, it is unclear how these additions would work with more complex experiments. In addition, these concepts may not seem biologically plausible. However, this domain-of-validity issue is a result of using linear interpolation to evaluate the f and g mappings. This potentially can be addressed by using other functional representations of the basis function such as the leaky integrate-and-fire model. These functions are non-zero over an infinite domain, and provide meaningful feedback even when values extend beyond the desired domain [12].

The next milestone for the SPE network is to create a working network for larger experiments. For instance, implementing some functionality of the visual cortex like Rao and Ballard's [30] experiment, or attempting some machine learning tasks such as digit recognition, a popular test for artificial neural networks.

Nonetheless, in this thesis, we laid out the framework for a variation on predictive estimators. We found that adding symmetry to a PE network allows for some functional advantages. Meanwhile, we discovered that SPEs are fit to be implemented into spiking neural models using the NEF. It is our hope that the concept and implementation of our SPE helps extend the theory of predictive coding and predictive estimators in the brain.

References

[1] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Roberto Lent, Suzana Herculano-Houzel, et al. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009.

[2] Andre M Bastos, W Martin Usrey, Rick A Adams, George R Mangun, Pascal Fries, and Karl J Friston. Canonical microcircuits for predictive coding. *Neuron*, 76(4):695–711, 2012.

[3] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7, 2013.

[4] Trevor Bekolay, Carter Kolbeck, and Chris Eliasmith. Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks. In *Proceedings of the 35th annual conference of the cognitive science society*, pages 169–174, 2013.

[5] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

[6] Horst Bischof, Werner Schneider, and Axel J Pinz. Multispectral classification of landsat-images using neural networks. *Geoscience and Remote Sensing, IEEE Transactions on*, 30(3):482–490, 1992.

[7] J Bullier, ME McCourt, and GH Henry. Physiological studies on the feedback connection to the striate cortex from cortical areas 18 and 19 of the cat. *Experimental Brain Research*, 70(1):90–98, 1988.

[8] Elise N Covic and S Murray Sherman. Synaptic properties of connections between the primary and secondary auditory cortices in mice. *Cerebral Cortex*, 21(11):2425–2441, 2011.

[9] Roberto De Pasquale and S Murray Sherman. Synaptic properties of corticocortical connections between the primary and secondary visual cortical areas in the mouse. *The Journal of Neuroscience*, 31(46):16494–16506, 2011.

[10] Julie Dethier, Vikash Gilja, Paul Nuyujukian, Shauki A Elassaad, Krishna V Shenoy, and Kwabena Boahen. Spiking neural network decoder for brain-machine interfaces. In *Neural Engineering (NER), 2011 5th International IEEE/EMBS Conference on*, pages 396–399. IEEE, 2011.

[11] Giuseppe Di Pellegrino, Luciano Fadiga, Leonardo Fogassi, Vittorio Gallese, and Giacomo Rizzolatti. Understanding motor events: a neurophysiological study. *Experimental brain research*, 91(1):176–180, 1992.

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

[13] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012.

[14] Apostolos P Georgopoulos, Andrew B Schwartz, and Ronald E Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986.

[15] James J Gibson. *The ecological approach to visual perception: classic edition.* Psychology Press, 2014.

[16] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63, 1987.

[17] David H Hubel, Janice Wensveen, and Bruce Wick. *Eye, brain, and vision.* Scientific American Library New York, 1995.

[18] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[19] Pierre Jacob and Marc Jeannerod. The motor theory of social cognition: a critique. *Trends in cognitive sciences*, 9(1):21–25, 2005.

[20] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *IEEE computer*, 29(3):31–44, 1996.

[21] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[22] James M Kilner, Karl J Friston, and Chris D Frith. Predictive coding: an account of the mirror neuron system. *Cognitive processing*, 8(3):159–166, 2007.

[23] Lindsay Kleeman. Understanding and applying kalman filtering. In *Proceedings of the Second Workshop on Perceptive Systems, Curtin University of Technology, Perth Western Australia (25-26 January 1996)*, 1996.

[24] Edward J Krakiwsky, Clyde B Harris, and Richard VC Wong. A kalman filter for integrating dead reckoning, map matching and gps positioning. In *Position Location and Navigation Symposium, 1988. Record. Navigation into the 21st Century. IEEE PLANS'88., IEEE*, pages 39–46. IEEE, 1988.

[25] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases*, pages 498–515. Springer, 2015.

[26] David MacNeil and Chris Eliasmith. Fine-tuning and the stability of recurrent neural networks. *PloS one*, 6(9):e22885, 2011.

[27] Marc Mignard and Joseph G Malpeli. Paths of information flow through visual cortex. *Science*, 251(4998):1249–1251, 1991.

[28] R. Kent. Nagle, E. B. Saff, and Arthur David. Snider. *Fundamentals of Differential Equations*. Pearson Education, 8 edition, 2011.

[29] Louis Albert Necker. Lxi. observations on some remarkable optical phænomena seen in switzerland; and on an optical phænomenon which occurs on viewing a figure of a crystal or geometrical solid. *Philosophical Magazine Series 3*, 1832.

[30] Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79–87, 1999.

[31] Giacomo Rizzolatti and Laila Craighero. The mirror-neuron system. *Annu. Rev. Neurosci.*, 27:169–192, 2004.

[32] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[33] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[34] Julie H Sandell and Peter H Schiller. Effect of cooling area 18 on striate cortex cells in the squirrel monkey. *J. Neurophysiol*, 48(1):38–48, 1982.

[35] S Murray Sherman and RW Guillery. On the actions that one nerve cell can have on another: distinguishing drivers from modulators. *Proceedings of the National Academy of Sciences*, 95(12):7121–7126, 1998.

APPENDICES

Appendix A

The Neural Engineering Framework

The brain is an incredibly complex organ and little is known about how the billions of neurons in the brain process information, interact with each other and perform complex cognition tasks. The Neural Engineering Framework (NEF) provides a base theory to build brain models and neural simulations that respect biological constraints [12]. The NEF is implemented by Nengo, a Python module that we used in this thesis to build the spiking neural models. The NEF is a versatile framework which has produced a large-scale brain model capable of performing diverse tasks such as digit recognition, motor control and pattern completion [13].

The three main principles of the NEF are neural representation, transformation and dynamics. These principles, alongside Nengo, allowed us to implement the SPE model into a biologically plausible network of spiking neurons.

A.1 Representation

A.1.1 Encoding

Biological experiments revealed that neurons have characteristic responses to stimulus called tuning curves [18]. The response is often stated as the number of spikes per second. Data in the NEF is represented by populations of many neurons with distinct tuning curves. A diverse group of neurons that respond to the same range of stimulus is called a neural ensemble. The combined neural activity from the ensemble can encompass data such as a scalar value or vector. This concept is quite prevalent in neuroscience literature [14].

In the NEF, the neural activity a_i of a neuron in response to a stimulus x, is governed by the encoding equation

$$a_i(x) = G_i\big[\alpha_i x \cdot e_i + J_i^{bias}\big], \tag{A.1}$$

where each neuron indexed by i, has a preferred response to a stimulus called an encoding vector e_i. The function G_i represents the neuron model such as the leaky-integrate-fire (LIF) neuron. The input to the G_i function represents the input current to the neuron, where G_i reflects the neuron's response to the current, either in the form of a firing rate or a time-series of spikes. The variable α_i represents a gain term, and J_i^{bias} represents a bias term. Note that the activity of the neuron will be negligible if e_i is not in the direction of neural input x. This equation calculates the activity of a neuron when holding the input current fixed.

A.1.2 Decoding

From (A.1) we are able to determine the neural activity to a stimulus x. But how does the NEF decode the value represented by the neural activity? This is known as neural decoding. The decoded value $\hat{x}$ is calculated by

$$\hat{x} = \sum_{i=1}^{N} a_i(x) d_i \tag{A.2}$$

where N is the number of neurons in the ensemble and d_i represents the decoding weights. This is simply a linear combination of neural activity $a_i(x)$ each multiplied by their respective decoding weight d_i. An example decoding result is illustrated in Figure A.1. The decoding weights are solved by minimizing the least squared error over the range of inputs X. We can sample the ensemble with this range of inputs to generate a matrix A, such that each column of the matrix represents the i^{th} neuron, and each row represents a discrete input from the representation range. The value stored in A is the neural activity in response to input X. The optimal values for decoders can be calculated by minimizing the error E as

$$E = \|AD - X\|^2 \tag{A.3}$$

where D represents the decoding weights in vector form. In addition, decoding weights can be found for a particular transformation f by substituting $f(X)$ for X.

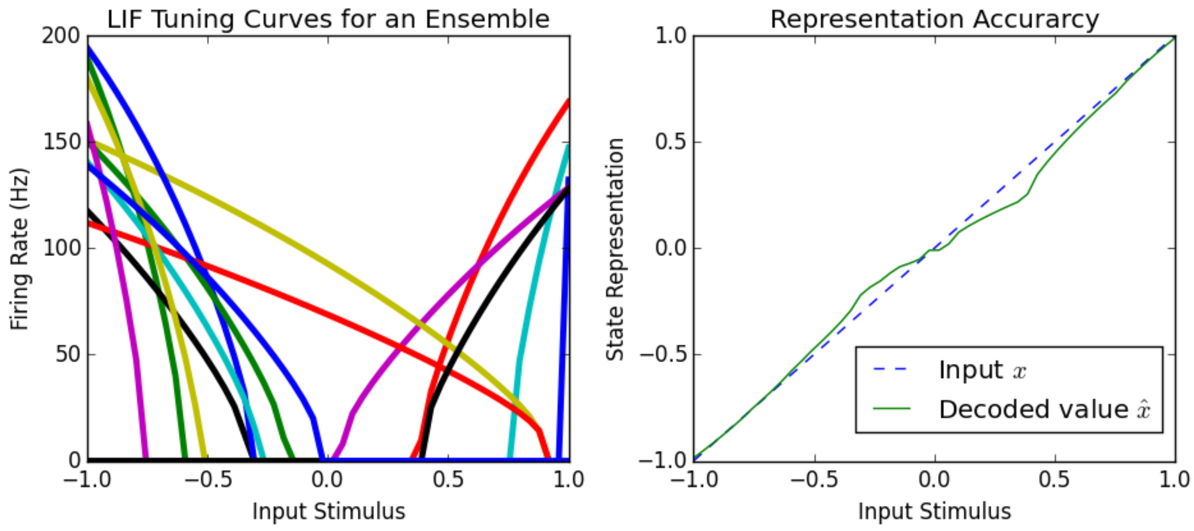


Figure A.1: Left: A population of neurons (ensemble) using the LIF model. Each neuron is distinguished by the colour of the tuning curve. The characteristics of each neuron is defined by their firing rate with respect the input stimulus. Right: A comparison of the decoded value $\hat{x}$ and the input x for an ensemble that represents the identity function. The decoded value is generated from a linear combination of decoders and the neural activity of the neurons in the ensemble.

A.1.3 Neuron Models and Temporal Representation

The LIF neuron can be used to represent the single cell model and function G_i. It is also the model that is used in our Nengo simulations. The membrane potential V, of the LIF neuron is goverened by the differential equation

$$\frac{dV}{dt} = \frac{1}{\tau_{RC}}\big[RJ - V\big] \tag{A.4}$$

where τ_{RC} is the neuron membrane time constant, R is the membrane resistance and J is the input current. The cell model allows us to gauge a neuron's firing rate given constant input current J. The model is governed by the equation

$$G_i(J) = \begin{cases} \frac{1}{\tau_{ref} - \tau_{RC} \ln\left(1 - \frac{J_{th}}{J}\right)} & J > J_{th}, \\ 0 & \text{otherwise,} \end{cases} \tag{A.5}$$

where τ_{ref} is the refractory period of the neuron, τ_{RC} is the neuron membrane time constant and J_{th} is the threshold voltage for the neuron. Figure A.1 shows the tuning curves for an example population of LIF neurons.

The premise of the LIF neuron is that when the membrane potential reaches a certain voltage via exchange of neurotransmitters, the neuron releases an action potential (spike) [12]. The NEF can perform temporal representation of neurons, in other words, time-dependent spikes instead of firing rates. When an LIF neuron spikes, we can think of its output as a series of as a time-shifted Dirac-delta functions, called a "spike train". When the spike train arrives at a synapse, it takes time for the signal to propagate to the next neuron. This process is modeled by convolving the spike train with a generic post-synaptic filter $h(t)$. Hence, the post-synaptic current induced from neuron i is proportional to $h(t) * a_i(x(t))$. By applying the decoding weights and summing up over all the neurons in an ensemble, it allows for the decoding of the dynamic value represented by the population of spiking neurons,

$$\hat{x}(t) = \sum_{i=1}^{N} h(t) * a_i(x(t))d_i \tag{A.6}$$

where $x(t)$ is the time-dependent input to the population. This allows the NEF to simulate the spike patterns of neurons often seen in biological data.

A.2 Transformation

Transformation describes the process of how distinct ensembles communicate data to each other. Given two populations of neurons P and Q, we would like a method to communicate data from P to Q and perform an arbitrary transformation. In a general sense, this involves decoding the value of the arbitrary function from P and using it to encode the value to Q. This process can be simplified as weights between ensembles

$$w_{ij} = \alpha_j e_j \cdot d_i \tag{A.7}$$

where i represents neuron indicies of P, j represents neuron indicies of Q and α_j represents gain terms for Q. Note that this is simply the outer product between the encoders e_j and the decoders d_j. This allows us to formulate the complete all-to-all weight matrix between encoders and decoders.

A.3 Dynamics

There are dynamics built into the NEF model that simulate the time dependencies of the connections between neurons. Like the real world, synapse connections between neurons are not instantaneous, and introduce a time-delay. This post-synaptic current is modeled by using a post-synaptic filter $h(t)$. These time-delays introduce some interesting dynamics.

A recurrent connection exists when the neural ensemble(s) are connected and form a cycle. A simple example is when one ensemble is connected to itself. In the NEF this introduces the ability to implement differential equations of the form of

$$\tau_s \frac{dx}{dt} = f(x(t)) - x(t), \tag{A.8}$$

where τ_s is a synaptic time constant. For instance, an integrator can be implemented such that the ensemble retains its value when the input is zero, and adds the input to the value when the input is non-zero. The equation of an integrator is modeled by

$$\frac{dx}{dt} = f(x(t)), \tag{A.9}$$

and can be implemented by having a recurrent connection with a transformation that outputs $\tau_s f(x(t)) + x(t)$. This recurrent connection is actually used in the SPE model for the state representation r_i. If the error values ε_i=0 and δ_i=0, the input values to r_i are zero and therefore retains its state representation value. Any non-zero error will subsequently change the state representation value.

Appendix B

End-stopping Phenomena in the Visual Cortex

Rao and Ballard performed an experiment that involved inputting natural images to a PE network [30]. Examples of natural images include forests, lakes and animals in their natural habitats. A majority of the images consisted of pixels that are strongly correlated in a dominant orientation (e.g. horizontal). For example, an image of trees had a strong vertical correlation of pixel intensities. Sampled image patches of 16 x 16 pixels provided bottom-up input to their network. The lower-level PE units were connected in a hierarchy such that the upper-level PE unit's image patch spanned three overlapping image patches (16 x 26).

While exposing the network to many different sample images, the PE network changed connection weights in order to minimize the residual error between PE layers. The reduction in residual error resulted in changes to connection weights that predicted mostly horizontal, vertical and diagonal lines.

They were able to use this result to explain biological phenomenon seen in the visual cortex. For example, a neuron in the visual cortex that is prone to exhibit neural activity only when the subject is exposed to horizontally oriented bars, is deemed a neuron that is tuned for horizontal stimulus. As the width of the horizontal bar increases, so does the observed spiking rates of the neuron. However, some neurons exhibit a near-zero spiking rate when the bar extends beyond the neuron's receptive field. This phenomenon is known as end-stopping and is shown in B.1 [17].

Rao and Ballard suggest there must be some external influence to the neuron that changes its behaviour. They suggest a theory that feed-back activity from the upper levels of a visual hierarchy inhibit the neural activity of the lower level. Explained using the PE model, groups of neighboring lower-level PE units receive a prediction from an upper-level unit. For example, assume the upper-level unit sends a prediction to the lower-level units that the input is a horizontal bar that spans the beyond width of the lower-level units. The residual error between the lower and upper-level units begin to decrease when a horizontal bar input begins to span the neighborhood of lower-level units. The reduction in neural activity is explained by a correct top-down prediction.

When the PE network was shown natural images after it was trained, the PE model generated similar neural responses to those seen in cortical neurons of a squirrel monkey [34]. The end-stopping phenomenon can be eliminated by freezing a higher layer of a squirrel monkey's visual cortex. A similar result occured when eliminating the feed-back from the PE model. This suggests that feed-back from the upper levels of the visual hierarchy play a role in the neural activity of lower-level neurons.

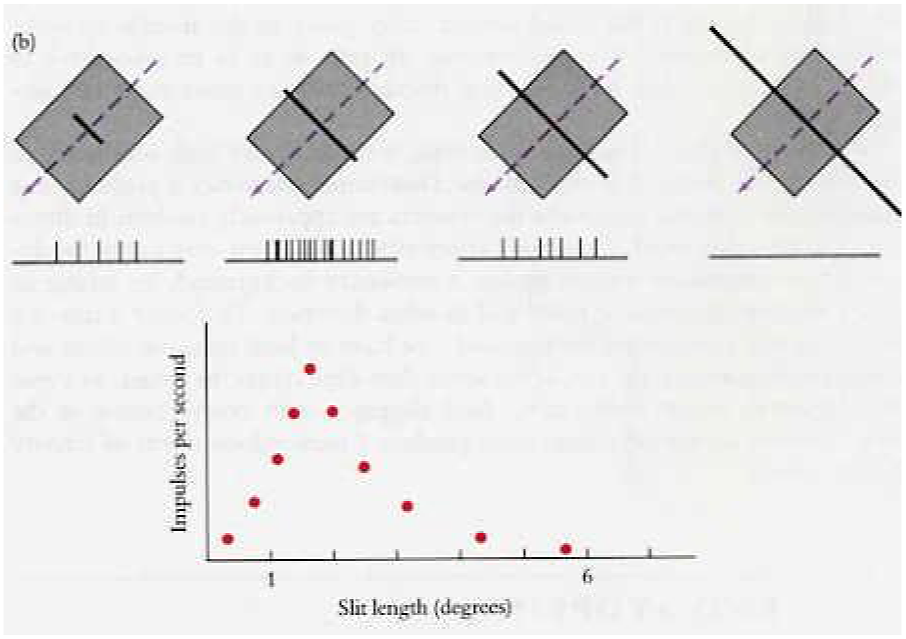


Figure B.1: Reproduced from figure in Chapter 4, Page 22 of [17]. Illustration of the end-stopping cells found in the visual cortex. When the slit of light (solid diagonal) exceeds the receptor field of the cell (grey rectangle), the firing rate is reduced to zero [17]. Rao and Ballard introduce a mechanism using predictive estimators to explain this phenomenon [30].