Chapter 4
Using Non-oscillatory Dynamics to Disambiguate Pattern Mixtures

Tsvi Achler

Abstract This chapter describes a model which takes advantage of the time domain through feedback iterations that improve recognition and help disambiguate pattern mixtures. This model belongs to a class of models called generative models.

It is based on the hypothesis that recognition centers of the brain reconstruct an internal copy of inputs using knowledge the brain has previously accumulated. Subsequently, it minimizes the error between the internal copy and the input from the environment.

This model is distinguished from other generative models which are unsupervised and represent early visual processing. This approach utilizes generative reconstruction in a supervised paradigm, implementing later sensory recognition processing.

This chapter shows how this paradigm takes advantage of the time domain, provides a convenient way to store recognition information, and avoids some of the difficulties associated with traditional feedforward classification models.

4.1 Introduction

To survive animals must interpret mixtures of patterns (scenes) quickly, for example, find the best escape path from an encircling predator pack; identify food in a cluttered forest. Scenes are commonly formed from combinations of previously learned patterns. Pattern mixtures are found in AI scenarios such as scene understanding, multiple voice recognition, and robot disambiguation of sensory information.

Difficulty in processing mixtures of patterns is a fundamental problem in connectionist networks. Difficulties are described even in the earliest connectionist literature as: "the binding problem" and "superposition catastrophe". Connectionist networks do not generalize well with pattern mixtures. As a consequence, the number of required nodes or training epochs grows exponentially as the number of patterns increases, causing a combinatorial explosion [6, 30, 32].

T. Achler (✉)
Siebel Center, University of Illinois Urbana-Champaign, 201 N. Goodwin Ave, Urbana, IL 61801, USA
e-mail: achler@illinois.edu

A.R. Rao, G.A. Cecchi (eds.), *The Relevance of the Time Domain to Neural Network Models*, Springer Series in Cognitive and Neural Systems 3,
DOI 10.1007/978-1-4614-0724-9_4,

To quantify the implications, let n represent the number of possible output nodes or representations. Let k represent the number of mixed patterns. The number of possible mixed pattern combinations is given by:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}. \tag{4.1}$$

If a network contains $n = 5{,}000$ patterns, there are about *12 million* two pattern combinations $k = 2$, and *21 billion* three pattern $k = 3$ combinations possible. If a network contains $n = 100{,}000$ patterns, there are *5 billion* $k = 2$ and *167 trillion* $k = 3$ combinations and so on. The brain must process mixed patterns more efficiently because it cannot train for each possible combination for the simple reason that there are not enough resources, such as neurons, let alone the time needed for such preparation.

Another way to quantify this difficulty is to compare the training and test distributions. Weight parameters are determined through the training data. Determination of weights requires the training distribution to be similar to the testing distribution. Similar distributions allow the correlation between input features and outcomes to be determined through a training set, and learning to occur. However, the training distribution is commonly violated in the natural (test) environment, such as a scene or overlapping patterns. If a network is trained on patterns A and B presented by themselves and they appear simultaneously, this is outside the training distribution. As previously discussed, training for every pair of possible patterns (or triplets, quadruplets, etc.) is combinatorially impractical.

It is often assumed that most combinatorial limitations can be overcome by isolating or segmenting patterns individually for recognition by utilizing their spatial location [11, 19–21, 24, 27, 28, 34, 38]. However, segmenting patterns in cluttered scenarios is difficult because prior recognition is often required for correct localization. Moreover, some modalities such as olfaction or taste do not allow for good spatial localization, because the sensory information is broadly scattered in space [10]. Yet emulating recognition performance in olfaction remains as difficult if not more difficult than spatial-rich modalities such as vision [8, 29].

Methods incorporating oscillatory dynamics, for example [31, 36] and this book, have been successful in differentiating components of patterns within a scene and partially addressing these problems. However, most methods require some a-priori segmentation.

This work does not make any assumptions about space, and patterns are not isolated or segmented in space. A dynamic model is proposed to address binding and pattern mixture processing issues that is independent of oscillations, synchronization, or spike timing. Yet it uses temporal dynamics. This Regulatory Feedback model focuses on the contribution of top-down inhibitory structures. It works by cycling activation between inputs and outputs. Inputs activate contending representations which in turn inhibit their own representative inputs. In general, inputs that are utilized by multiple representations are more ambiguous (not as unique) compared to those utilized by single representations. Ambiguous inputs are inhibited by multiple outputs and become less relevant. The inhibited inputs then affect representation activity, which again affects inputs. The cycling is repeated until a steady

state is reached. This method facilitates simultaneous evaluation of representations and determines sets of representations that best fit the whole "scene". The implementation of non-oscillatory feedback dynamics for separating patterns is described in detail and key examples are demonstrated by simulations.

4.1.1 Evidence for Top-Down Feedback Regulation in the Brain

Regulation of an input by the outputs it activates is referred to as regulatory feedback. Sensory processing regions of the brain (e.g. thalamus, olfactory bulb, sensory cortex) have a massive amount of reentrant top-down feedback pathways which regulate bottom-up processing. Furthermore, a large variety of brain mechanisms can be found where inputs are regulated, even at the level of synapses [9, 15, 18].

Anatomical studies provide evidence for top-down regulatory connections. For example, in the thalamo-cortical axis, thalamic relay neurons (inputs) innervate cortical pyramidal cells (outputs). These cortical neurons feed-back and innervate the thalamic reticular nucleus. The neurons from the thalamic reticular nucleus inhibit the relay neurons. Feedback must be very tightly controlled because there are more feed-back connections than feed-forward connections [13, 14, 16, 22, 23].

Homeostatic plasticity provides another form of evidence. In homeostatic plasticity, pre-synaptic cells (inputs) will change their activation properties to restore homeostasis (fixed input to output activity levels), between pre- and post-synaptic cells (between inputs and outputs). This requires top-down (output to input) regulatory feedback targeting pre-synaptic neurons [25, 35]. Although the timescale is slower, homeostatic plasticity provides evidence that connections required for regulatory feedback exist. The real time contribution of such connections can facilitate conservation of information.

4.1.2 Difficulties with Large Scale Top-Down Regulatory Feedback

Unfortunately, large-scale amounts of regulatory feedback are difficult to incorporate into models, because the mathematics become highly nonlinear and difficult to analyze. Control theory, a field in engineering, is dedicated to the study of feedback properties. Within this field, regulatory feedback would be considered to as a type of negative feedback. However, during recognition, most neural models focus on more tractable configurations, such as parameterized feed-forward connections and lateral inhibition (see Fig. 4.1).

4.2 Regulatory Feedback Networks

The feedback regulation algorithm is realized as a two layer network with fuzzy-type input features x, output nodes y, and no hidden units. Regulatory feedback is demonstrated here using nodes with binary connections to better appreciate the contribution of dynamics. The point of implementing binary connections is to appre-

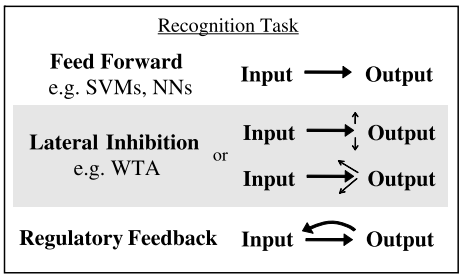

Fig. 4.1 Comparison of mechanisms utilized in recognition. Feedforward connections: neural networks and support vector machines. Lateral inhibition: winner-take-all. Feedback regulation: top-down self-inhibition, negative feedback

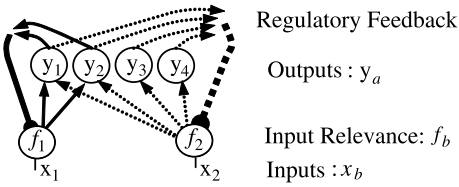

Fig. 4.2 Example configuration of symmetric self-regulation. If x_1 activates $y_1 \& y_2$, then feedback from $y_1 \& y_2$ regulates x_1. Similarly, if x_2 activates y_1, y_2, $y_3 \& y_4$, then feedback from y_1, y_2, $y_3 \& y_4$ regulates x_2. Thus, for this example: $M_1 = \{y_1, y_2\}$, $M_2 = \{y_1, y_2, y_3, y_4\}$, $G_1 = G_2 = \{x_1, x_2\}$, $G_3 = G_4 = \{x_2\}$, $g_3 = g_4 = 1$, $g_1 = g_2 = 2$

ciate limits of feedforward networks with analagous binary connections. Dynamic recognition is possible even if all input features are connected equally to associated output nodes (in a binary fashion). In regulatory feedback networks input relevance f is determined at the time of recognition; for example, value $f = 1$ is salient, but 2 is twice as salient and 0.5 is half.

Each output node y_a is defined by set of feedforward binary connections G_a from input x's. It also has a set of symmetrical feedback connections M_a that implement negative feedback, modulating relevance (Fig. 4.2). Connectivity strengths are uniform here, which means all x's that project to a y have the same weight. Similarly all of a y's feedback connections to its x's have the same weight. Let's label M_b the connections that return to an input x_b. Let's label f_b the relevance value of input x_b. Then the activity of the output node is determined by:

$$y_a(t + dt) = \frac{y_a(t) \cdot dt}{g_a} \sum_{i \in G_a} f_i \tag{4.2}$$

with regulatory feedback affecting every input b

$$f_b = \frac{x_b}{Y_b} \tag{4.3}$$

through feedback term.

$$Y_b = \sum_{j \in M_b} y_j(t). \tag{4.4}$$

For an output node y_a, set G_a denotes all its input connections and g_a denotes the number of these connections. For an input x_b, set M_b denotes all output nodes

connected to it. The feedback from output nodes to input x_b is Y_b. The value Y_b represents the sum of all of the output nodes that use x_b. The input relevance is f_b, determined by feedback composed of the sum of all the outputs using that input.

Feedback modifies relevance values. The modified relevance values are redistributed to the network, determining new output activations. This modification process is repeated iteratively to dynamically determine final relevance of inputs and recognition. The iterative nature allows inference during the recognition phase.

The networks dynamically test recognition of representations by (1) evaluating outputs based on inputs and their relevance, (2) modifying the next state of input relevance based on output use of the inputs, (3) reevaluating outputs based on the new input relevance. Steps 1–3 represent a cycle which can be iteratively repeated.

Modularity and Scalability This structure allows modularity and scalability because each node only connects to its own inputs. Thus, output nodes do not directly connect with the other output nodes. Instead they interact indirectly through shared inputs. As a result, an addition of a new node to the network only requires that it forms symmetrical connections to its inputs (an input-to-output connection paired with an equivalent output-to-input connection). Subsequently, the number of connections of a node is independent of the size or composition of the network. Yet the networks make complex recognition decisions based on distributed processing, as demonstrated in this work.

4.2.1 Description of Function

The networks dynamically balance the ratio of input and output of activity. Suppose an input provides information x_a to its outputs Y_a. These outputs feed back in unison to the input x_a and regulate its relevance in a 'shunting' fashion $f_a = x_a / Y_a$, whereby inhibition does not completely eliminate an input's activity but reduces it. The tight association creates a situation where the information x_a can be fully expressed to the output layer only if $Y_a = 1$ (which occurs when inputs and outputs are matched). If several outputs are overly active, no output will receive the full activity of x_a because $Y_a > 1$ thus $f_a < x_a$. Conversely if x_a is not appropriately represented by the network then $Y_a < 1$ and the input is boosted $f_a > x_a$. In that case outputs will recieve more input activity than designated by x_a. This regulation occurs for every input–output interaction.

Additionally, each output node strives for a total activation of 1. Thus, if an output cell has N inputs each connection contributes $1/N$ activity. This normalizes output activity and promotes the relevance value of 1 as a steady state fixed point.

4.2.2 General Limits, Stability, Steady State, & Simulations

The outputs y are shown analytically to be bounded between zero and x values [1, 4] and this class of equations settle to a non-oscillatory steady state equilibrium [26].

In this network, all variables are limited to positive values. Thus, the values of y cannot become negative and have a lower bound of 0. The upper values of y are bounded by the maximal input given value $x_{\max}$. y_a activity will be greatest if its relevance f_{G_a} is maximized. Relevance will be maximized if nodes that share its inputs are not active $M_a \notin y_a = 0$. Assuming this is the case then the equation simplifies to:

$$\begin{aligned} y_a(t+dt) &= \frac{y_a(t)}{g_a} \sum_{i \in G_a} f_i \\ &\leq \frac{y_a(t)}{g_a} \sum_{i \in G_a} \left(\frac{x_{\max}}{y_i(t)+0} \right) \\ &= \frac{1}{g_a} \sum_{i \in G_a} (x_{\max}) = \frac{x_{\max} \cdot g_a}{g_a} = x_{\max}. \end{aligned}$$

If $x_{\max}$ is bounded by 1, then y_a is bounded by 1. The values are bounded by positive numbers between zero and $x_{\max}$, satisfying boundary conditions [6]. Furthermore as $dt \to 0$, a Lyapunov function can be written. This indicates that the networks will settle to a steady state and not display chaotic oscillations. Numerical simulations also show the equations are well behaved.

In numerical simulations, y_a is constrained to a minimum of epsilon ε to avoid a divide-by-zero error (ε is a very small real number; 10^{-7} in practice). Steady state is defined when all output node values change by less than 0.0001 in a simulation cycle.

4.2.3 Simple Two Node Demonstration

A simple toy two node configuration (Fig. 4.3, Example 1) is presented to illustrate the workings and dynamics of the model. Two nodes are connected such that the first node y_1 has one input (x_1) which overlaps with the second node y_2. y_2 has two inputs ($x_1 \& x_2$). Since y_1 has one input, $g_1 = 1$ in the equation for y_1. Since y_2 has two inputs $g_2 = 2$ in the equation for y_2 and each input to y_2 contributes one-half. x_1 projects to both $y_1 \& y_2$, thus f_1 receives feedback from both $y_1 \& y_2$. x_2 projects only to y_2 so f_2 receives feedback only from y_2.

Such compositions can be created in a modular fashion. Suppose one node represents the pattern associated with the letter P and another node represents the pattern associated with the letter R. R shares some inputs with P. These nodes can be combined into one network, by just connecting the network and ignoring this overlap. This defines a functioning network without formally learning how nodes R and P should interact with each other. Whether the features in common with R and P should predominantly support R or P or both is determined dynamically during testing (via activation and feedback), not training (i.e., though an a-priori predetermined weight).

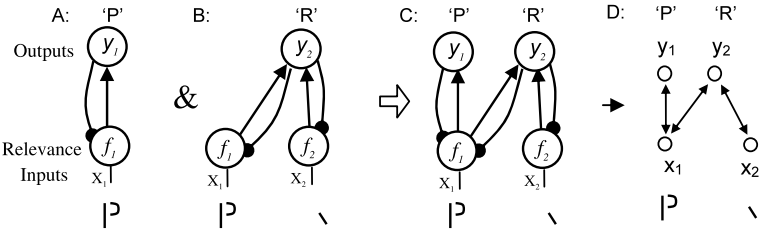

Fig. 4.3 (**A–C**), Example 1: Modular nodes y_1 and y_2 (**A** & **B**, respectively) can be simply combined to form a coordinated network (**C**). No modifications to the base nodes or new connections are needed. The nodes interact with each other at the common input, x_1. Since the feedforward and feedback connections are symmetric, the network can be drawn using bidirectional connections (**D**)

Equations of the above configuration are given by:

$$y_1(t+dt) = \frac{y_1(t)x_1}{y_1(t)+y_2(t)}, \tag{4.5}$$

$$y_2(t+dt) = \frac{y_2(t)}{2}\left(\frac{x_1}{y_1(t)+y_2(t)} + \frac{x_2}{y_2(t)}\right). \tag{4.6}$$

Stepping through iterations yields an intuition about the algorithm, see Fig. 4.4. Lets assume $x_1 = x_2 = 1$, with the initial activity of y_1 and y_2 assumed to be 0.01 ($y_1 = 0.01$ and $y_2 = 0.01$ at $t < 1$). $t = 0$ represents the initial condition. At $t = 1$ *backward* the activity of $y_1 \& y_2$ are projected back to the inputs. Both s_1 and s_2 are boosted because representations that use $x_1 \& x_2$ are not very active (initial values are 0.01). Thus, the input relevance (f_1 and f_2) of $x_1 \& x_2$ are boosted. Note that f_2 is boosted more than f_1 because two nodes use input f_1. At $t = 1$ *forward* the inputs modulated by salience are projected to the output nodes. Note that both y_1 and y_2 gain activation. The new activation of the output node is a function of the node's previous activity and the activity of the inputs normalized by the number of node processes (g_a). At $t = 2$ *backward* the activity of $y_1 \& y_2$ are projected back to the inputs again. This time the output nodes are more active so $f_1 \& f_2$ values are smaller. From $t = 2$ *forward* to $t \to \infty$ this trend continues, reducing y_1 activity while increasing y_2 activity. At $t = \infty$, the steady state values of y_1 becomes 0 and y_2 becomes 1.

The solutions at steady state can be derived for this simple example by setting $y_1(t+dt) = y_1(t)$ and $y_2(t+dt) = y_2(t)$ and solving the equations. The solutions are presented as (*input values*) $\to$ (*output vectors*) where $(x_1, x_2) \to (y_1, y_2)$. The solutions are $(x_1, x_2) \to (x_1 - x_2, x_2)$. The network cannot have negative values (see section on limits and stability) thus if $x_1 \leq x_2$ then $y_1 = 0$ and the equation for y_2 becomes: $y_2 = \frac{x_1+x_2}{2}$. The results for $x_1 = x_2 = 1$ is $(1, 1) \to (0, 1)$. If $x_1 = 1$ and $x_2 = 0$, then y_1 is matched: $(1, 0) \to (1, 0)$.

Thus, if inputs 'P' & '\' are active y_2 wins. This occurs because when both inputs are active, y_1 must compete for all of its inputs with y_2, however y_2 only needs to compete for half of its inputs (the input shared with y_1) and it gets the other half 'free'. This allows y_2 to build up more activity and in doing so inhibit y_1.

Thus, smaller representation completely encompassed by a larger representation becomes inhibited when the inputs of the larger one are present. The smaller repre-

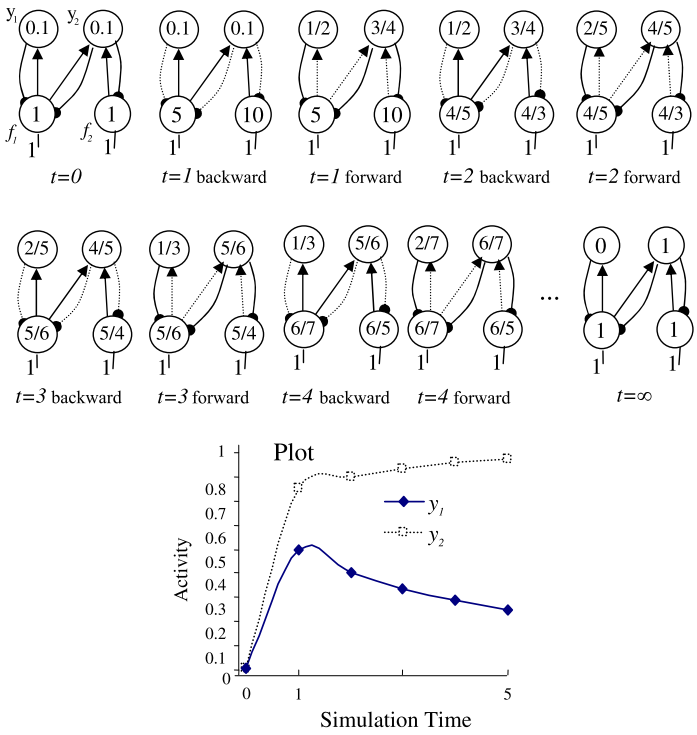

Fig. 4.4 Example 1: Numerical simulation. *Top*: steps in simulation and intermediate values. $t = 0$ initial condition; $t = 1$ feedback contribution at time 1; $t = 1$ feed forward contribution at time 1; $t = 2$ backward next cycle feedback; $t = \infty$ steady state. *Bottom Right*: y_1&y_2 value plot for first five cycles

sentation is unlikely given features specific only to the large representation. A negative association was implied ('y_1' is unlikely given feature 'x_2') even though it was not directly encoded by training. However, in feedforward networks, such associations must be explicitly trained as demonstrated in Example 2, Fig. 4.8. Such training requires the possible combinations to be present in the training set, resulting in a combinatorial explosion in training. The Binding and Superposition examples demonstrate these training difficulties.

Lastly, though integer inputs are presented, the solutions are defined for positive real x input values and the system is sensitive to numerosity [7]. With twice as much input activity there is twice as much output activity: $(2, 2) \rightarrow (0, 2)$. Patterns are also recognized as mixtures: $(2, 1) \rightarrow (1, 1)$.

4.3 The Binding Problem Demonstration

The binding problem occurs when image features can be interpreted through more than one representation [31, 36]. An intuitive way to describe this problem is

Fig. 4.5 Illusion representing a linearly dependent scenario. In this illusion, multiple labels exist for the same pattern. Every feature can either be interpreted as part of an old women or young woman

through certain visual illusions. In the old woman/young woman illusion of Fig. 4.5, the young woman's cheek is the old woman's nose. A local feature can support different representations based on the overall interpretation of the picture. Though the features are exactly the same, the interpretation is different. In humans, this figure forms an illusion because all features in the image can fit into two representations. Classifiers have similar difficulties but with simpler patterns. If a pattern can be part of two representations, then the networks must determine to which it belongs. Training is used to find optimal interconnection weights for each possible scenario. However, this is not trivial: combinations of patterns and training can grow exponentially.

A simple scenario is presented to show that the regulatory feedback equations behave in a manner which is beneficial towards binding. This scenario demonstrates that compositions of partially overlapping representations can cooperate and/or compete to best represent the input patterns. The equations allow two representations to inhibit a third. The performance of several networks are compared to better illustrate the binding problem within this scenario.

In this composition, three representations are evaluated simultaneously. Example 2 Fig. 4.6 is expanded from Example 1 Fig. 4.3, with a modular addition of a new output node y_3. As in Example 1, y_1 competes for its single input with y_2. However, now y_2 competes for its other input with y_3 and y_3 competes for only one of its inputs.

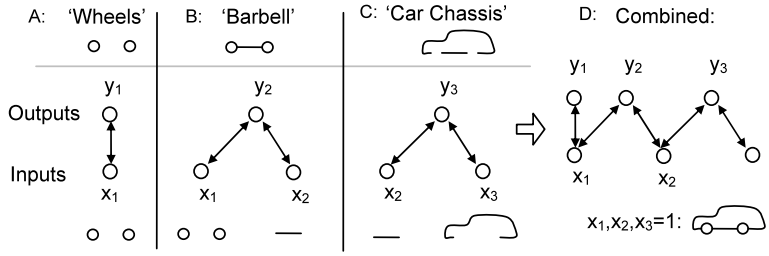

Fig. 4.6 (**A–D**): Modular combination of nodes display binding. Nodes y_1, y_2, y_3 (**A**, **B** & **C**) can be simply combined to form a combined network (**D**). Yet these patterns interact in cooperative and competitive manner finding the most efficient configuration with the least amount of overlap

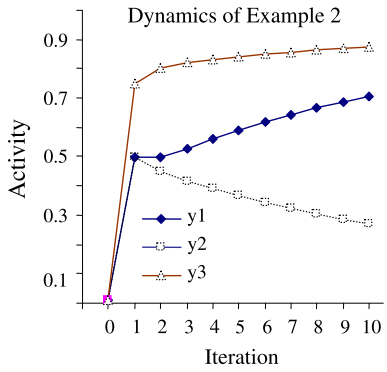

Fig. 4.7 Simulation of Example 2. Graph of y_1, y_2, & y_3 dynamics given car (1, 1, 1)

Equation for y_1 remains the same as Example 1. Equations for y_2 and y_3 become:

$$y_2(t+dt) = \frac{y_2(t)}{2}\left(\frac{x_1}{y_1(t)+y_2(t)} + \frac{x_2}{y_2(t)+y_3(t)}\right), \tag{4.7}$$

$$y_3(t+dt) = \frac{y_3(t)}{2}\left(\frac{x_2}{y_2(t)+y_3(t)} + \frac{x_3}{y_3(t)}\right). \tag{4.8}$$

Solving for steady state by setting $y_1(t+dt) = y_1(t)$, $y_2(t+dt) = y_2(t)$, and $y_3(t+dt) = y_3(t)$, the solutions are $y_1 = x_1 - x_2 + x_3$, $y_2 = x_2 - x_3$, $y_3 = x_3$. Thus, $(x_1, x_2, x_3) \rightarrow (y_1 = x_1 - x_2 + x_3, y_2 = x_2 - x_3, y_3 = x_3)$. If $x_3 = 0$ the solution becomes that of for example, 1: $y_1 = x_1 - x_2$ and $y_2 = x_2$. If $x_2 \leq x_3$, then $y_2 = 0$ and the equations become $y_1 = x_1$ and $y_3 = \frac{x_2+x_3}{2}$.

Solutions to particular input activations are:

$$(1,0,0) \rightarrow (1,0,0); \qquad (1,1,0) \rightarrow (0,1,0); \qquad (1,1,1) \rightarrow (1,0,1).$$

If only input x_1 is active, y_1 wins. If only inputs x_1 and x_2 are active y_2 wins for the same reasons this occurs in Example 1. However, if inputs x_1, x_2 and x_3 are active then y_1 and y_3 win. The network as a whole chooses the cell or cells that best represent the input pattern with the least amount of competitive overlap.

The dynamics of this network are plotted in Fig. 4.7. In this configuration, y_2 must compete for all of its inputs: x_1 with y_1, x_2 with y_3. y_3 only competes for half of its inputs (input x_2) getting input x_3 'free'. Since y_2 is not getting its other input x_1 'free' it is at a competitive disadvantage to y_3. Together y_1 and y_3, mutually benefit from each other and force y_2 out of competition. Competitive information travels indirectly 'through' the representations. Given active inputs x_1 and $x_2 = 1$, the activity state of y_1 is determined by input x_3 through y_3. If input x_3 is 0, then y_1 becomes inactive. If input x_3 is 1, y_1 becomes active. However, y_3 does not share input x_1 with y_1.

Suppose the inputs represent spatially invariant features where feature x_1 represents circles, x_3 represents the body shape and feature x_2 represents a horizontal bar. y_1 is assigned to represent wheels and thus when it is active, feature x_1 is interpreted as wheels. y_2 represents a barbell O–O composed of a bar adjacent to two round weights (features x_1 and x_2). Note: even though y_2 includes circles (feature

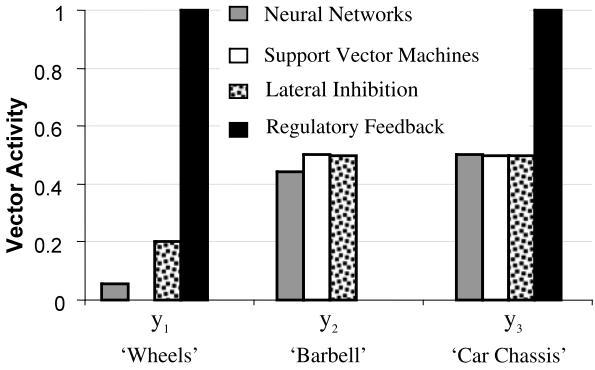

Fig. 4.8 Binding Problem and Classifier Responses given car (1, 1, 1). Of recognition algorithms applied, only regulatory feedback determines solution $Y_1 \& Y_3$ which covers all inputs with the least amount of overlap between representations. Other methods must be further trained to account for the specific binding scenarios

x_1), they do not represent wheels (y_1), they represent barbell weights. Thus, if y_2 is active feature x_1 is interpreted as part of the barbell. y_3 represents a car body without wheels (features x_2 and x_3), where feature x_2 is interpreted as part of the chassis. Now given an image of a car with all features simultaneously (x_1, x_2 and x_3), choosing the barbell (y_2) even though technically a correct representation, is equivalent to a binding error within the wrong context in light of all of the inputs. Thus choosing y_2 given (1, 1, 1)  is equivalent to choosing the irrelevant features for binding. Most classifiers if not trained otherwise are as likely to choose barbell or car chassis (see Fig. 4.8). In that case, the complete picture is not analyzed in terms of the best fit given all of the information present. Similar to case 1, the most encompassing representations mutually predominate without any special mechanism to adjust the weighting scheme. Thus, the networks are able to evaluate and bind representations in a sensible manner for these triple cell combinations.

Similar to the illusion in Fig. 4.5, multiple representations cooperate to interpret the picture. In the illusion, all of the features either are interpreted as a component of either representation (but not confused as a feature of any other representations).

NN and SVM algorithms are tested using the publicly available *Waikato Environment for Knowledge Analysis* (WEKA) package [37]. The purpose of the WEKA package is to facilitate algorithm comparison and maintains up to date algorithms (2007). The *lateral inhibition* and *feedback regulation* algorithms utilize identical feed-forward connections, but in *lateral inhibition* output nodes compete in a winner-take-all fashion and their fit ranked. Lateral Inhibition combined with ranking of fit (Sect. 4.4) is similar to the Adaptive Resonance Theory (ART) [12].

4.4 Superposition Catastrophe Demonstration

Superposition Catastrophe can be demonstrated in scene-like scenarios, where patterns are presented together without a good way to separate them. The presented toy example demonstrates this problem and how regulatory feedback overcomes it. Like the binding scenario, algorithms are all given the same supervised training set and

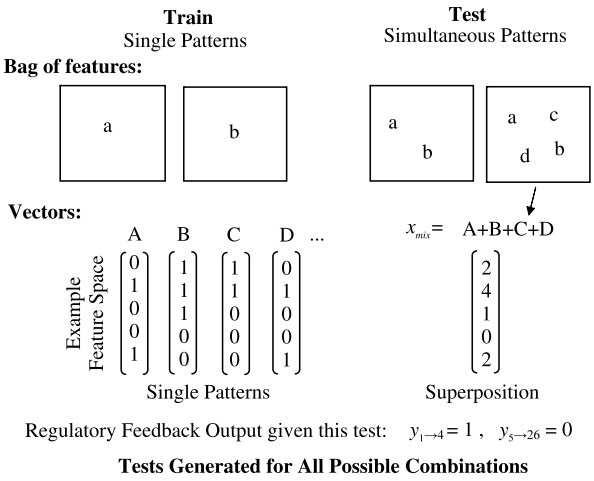

Fig. 4.9 Training with single patterns, testing with pattern mixtures. Test patterns are a superposition mixture of multiple trained patterns. *Top*: Emulation demonstrating mixtures of patterns presented in a bag-of-features feature extractor. *Bottom*: Representative vectors of the patterns. Only the first 5 inputs features of the feature space are shown

test cases. In this case, the training set is composed of randomly generated vector patterns with binary connections. Each input feature has a 50% probability of being associated to each node. There are 512 possible input features. Let n represent the number of possible output nodes. In the examples presented, $n = 26$. These numbers are arbitrary but only 26 random outputs are defined so the output nodes and associated vectors can be labeled $\mathbf{A} \rightarrow \mathbf{Z}$. Networks are trained on single vectors (**A**..**Z**) but tested on simultaneous vectors.

To recognize the 26 patterns, one output node (y) is designated for each pattern. Network "training" is rudimentary. During learning, single patterns are presented to the network, this activates a set of feature inputs. The output node is simply connected to every feature input that is active. Thus, each respective output node is designated to a pattern and is connected to all input features associated to that pattern.

A superposition of pattern mixtures is achieved by adding features of several vectors together generating an intermixed input vector, $\mathbf{x}_{\text{mix}}$ (see Fig. 4.9 bottom). Let $\mathbf{x}_{\text{mix}}$ represent inputs of pattern mixtures superimposed. For example, $\mathbf{x}_{\text{mix}} = \mathbf{A} + \mathbf{B}$ is composed of the sum vectors **A** and **B**. If vectors **A** and **B** share the same feature, that feature's amplitude in $\mathbf{x}_{\text{mix}}$ is the sum of both. This is analogous to a scene composed of multiple objects or an odorant mixture.

One way to implement this is through a bag-of-features type feature extractor or convolution applied to all stimuli in the visual field. Such extractors are similar in spirit to feature extraction found in the primary visual cortex V1, and commonly implemented in cognitive models [33]. $\mathbf{x}_{\text{mix}}$ would be the result of placing multiple patterns (far apart) within the bag-of-features extractor (see Fig. 4.9, top).

Networks that are only trained with single patterns are evaluated on scenes composed of two and four pattern mixtures. There are 325 possible tests (e.g., $\mathbf{x}_{\text{mix}} = \mathbf{A} + \mathbf{B}$, $\mathbf{A} + \mathbf{C}$, $\mathbf{A} + \mathbf{D} \ldots$) with two pattern mixtures, $k = 2$, $n = 26$, see Eq. 4.1. There are 14,950 possible tests (i.e., $\mathbf{x}_{\text{mix}} = \mathbf{A}+\mathbf{B}+\mathbf{C}+\mathbf{D}$, $\mathbf{A}+\mathbf{B}+\mathbf{D}+\mathbf{G}$,

$\mathbf{A}+\mathbf{B}+\mathbf{E}+\mathbf{Q},\ldots$) with four simultaneous patterns, $k=4$. All combinations are presented to the network.

At the end the test for a specific combination, the highest ky-values are chosen and their identity is compared to the patterns in $\mathbf{x}_{\text{mix}}$. If the top nodes match the patterns that were used to compose the $\mathbf{x}_{\text{mix}}$, then the appropriate histogram bin (*#number of patterns_matched/k*) is incremented. Suppose $k=4$, and for a specific test $\mathbf{x}_{\text{mix}}=\mathbf{A}+\mathbf{B}+\mathbf{C}+\mathbf{D}$. If the 4 top active nodes are A, B, C, G, then only three of the top four nodes match the composition in $\mathbf{x}_{\text{mix}}$. The count of bin 3/4 will be incremented.

4.4.1 Results

Neural Networks, Support Vector Machines, and lateral inhibition are again tested. Each method is given single prototypical representations (features of single patterns) for the training phase and tested on multiple representations (features of multiple patterns summed and presented to the network). Only Regulatory Feedback Networks correctly recognize the mixtures (tested up to eight pattern mixtures and score 100%). This occurs despite nonlinear properties of regulatory feedback networks. Regulatory feedback does not require the training and test distributions to be similar [6].

If $k=4$ (Fig. 4.10 bottom), there are 14,900 possible combinations 4 pattern mixtures. In order for NN, SVMs, and lateral inhibition to match accuracy of feedback inhibition training on the $\sim 15k$ combinations may be required. This is impractical because such numbers grow exponentially with the number of patterns as indicated by Eq. 4.1.

These tests were repeated with various randomly generated patterns formed with various n's (up to 100) and k's (up to 8). As long as (1) no two representations are the same or (2) two representations do not compose a third, regulatory feedback recognized all combinations with 100% success.

4.5 Discussion

These results suggest some of the difficulties within the Superposition Catastrophe and The Binding Problems may originate from the static nature of feedforward models. Dynamic networks allow more robust processing of scenes. This in turn reduces the reliance on scene segmentation while providing more information in order to perform better scene segmentation.

Though combinatorial problems have been associated with feedforward models [30, 32], the feedforward model remains popular because of its flexibility with large parameter spaces.

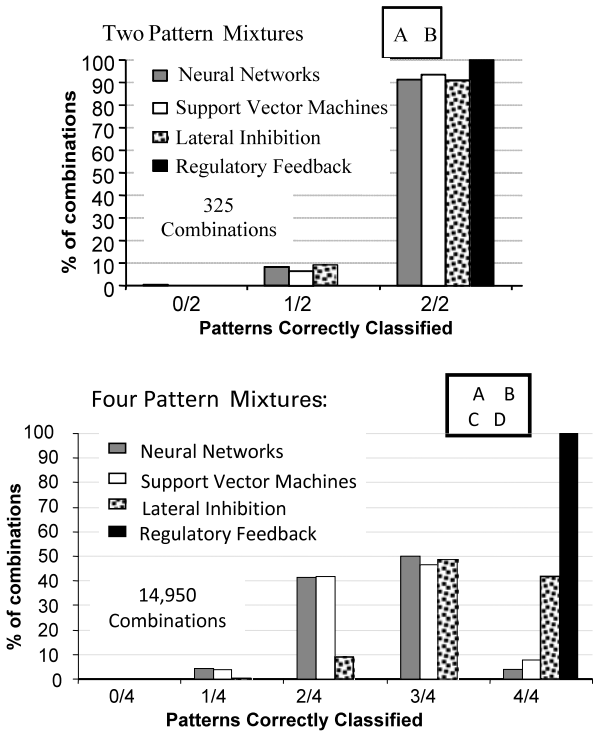

Fig. 4.10 Superposition catastrophe demonstration. Each network is trained on the same single patterns and tested on the same combinations of the patterns. All possible pattern combinations are tested. The top k active nodes of each network are selected and compared to the presence of original stimuli. The appropriate number_correct/k histogram bin is incremented. *Top*: Results for two pattern mixtures. *Bottom*: Results for four pattern mixtures

4.5.1 Plasticity Due to Dynamics in Regulatory Feedback Networks

In this demonstration, the patterns **A**..**Z** are linearly dependent. This means that they cannot be expressed as combinations of each other. However if linear dependence does occur regulatory feedback networks still function in a rational manner. For example, if $\mathbf{A} = \mathbf{B}$ (see Fig. 4.11 Case I) then y_A and y_B have same inputs (two labels sharing the same inputs). If that occurs, the network matches the base pattern inputs (vector **A**) and distributes the activation to the dependent patterns. Thus, at steady state $y_A + y_B$ will equal to activation garnered by the **A** configuration. Whether $y_A > y_B$, $y_A < y_B$, or $y_A = y_B$ will be determined by initial values of y_A and y_B. This is a semi-plastic regime that maintains memory of the network's previous state. This affects the dependent representations only. For example, with independent $\mathbf{C} \rightarrow \mathbf{Z}$ vectors the network correctly determines mixtures that do not include **A** or **B**. If **A** or **B** are present in the mixture, then there is 50% chance of being correct (depends on initial conditions which are randomized). Dependent label conflicts can also occur with combinations of representations. For example, if $\mathbf{C}+\mathbf{D} = \mathbf{E}$ (Case II) then activation of **E** will be distributed to nodes y_C, y_D, and y_E. The network's responses to dependent representations are logical because the network does not have a way to differentiate between the dependent representations. However, learning algorithms should strive to avoid linear dependent representations so that recognition can be unambiguous. One approach to avoiding linear de-

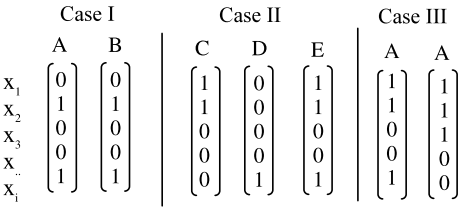

Fig. 4.11 Types of degenerate training cases. Case I: same vectors with different labels. Case II: Two vectors equivalent to a third. Case III: Different vectors with the same label, commonly presented in a traditional learning scenario

pendence is to introduce hierarchical layers to the network whenever independence is violated [3].

However, if recognition information is sufficiently impoverished or incomplete, and only reveals features common to two patterns, or is an illusion such as Fig. 4.5, this can also place the network in the linear dependence state. The network cannot decide between multiple equal solutions. In such cases, network response depends on initial conditions: a semi-plastic regime that maintains memory of the network's previous state. This is a rational strategy to address incomplete information.

4.5.1.1 Plasticity and Neuroscience Implications

Neuroscience experiments that are commonly interpreted as a test for memory place feedback networks in the linear dependence regime. This semi-plastic regime is also a feedback alternative to the synaptic plasticity hypothesis. Commonly neuroscience experiments are cited as evidence of parameter optimization through the idea of Hebbian synaptic plasticity. Synaptic plasticity in this context of learning is the most studied phenomenon in neuroscience. Synaptic plasticity is assumed to occur in sensory regions, whenever a long-lasting change in communication between two neurons occurs as a consequence of stimulating them simultaneously. However, the mechanisms that control activity-dependent synaptic plasticity remain unclear because such experiments are variable [5, 17]. Furthermore, regulatory feedback networks can emulate phenomena seen in experiments (without synaptic plasticity), suggesting an alternate explanation for such learning [2].

Clearly, robust learning occurs in the brain. However the underlying mechanisms for plasticity (e.g., Spike timing dependent plasticity) may involve dynamics.

4.5.1.2 Non-binary Connection Weights

The focus of this chapter is to demonstrate the dynamics of this network and its ability to recognize combinations of patterns. For simplicity and to better demonstrate the plasticity mechanism due to dynamics, I focused on binary connections and learning the connectivity was trivial. Connectivity can be extended to weighted connections, however this is outside the scope of the chapter.

4.5.2 Measuring Difficulty Processing

The difficulty of mixture pattern processing can be evaluated by the number of cycles required to reach steady state. The greater number of cycles required the harder the task. The number of iterations needed increased linearly with the number of patterns mixed. It took an average of 23 cycles in the 1 pattern case, 42 cycles in the $k = 2$ mixture pattern case, and 82 cycles in the $k = 4$ pattern case to reach steady state. It is important to note that the difficulty of this task as measured by cycle time increases only linearly with the number of combinations even while the number of combinations increase exponentially.

4.5.2.1 Notes on Cycles

The number of iterations depends on the definition of steady state and the value of *dt* utilized in the equation. The number of iterations necessary can vary based on a speed accuracy trade-off. To be very accurate, the criteria for steady state (and *dt*) should be small. *dt* remained 1 throughout all simulations and steady state was declared when the sum of y's do not vary more than 0.0001 between cycles. To increase speed, the criteria for steady state can be increased.

4.5.3 Combined Oscillatory and Non-oscillatory Dynamics

Like feedforward methods, this method does not preclude additional oscillatory units to enhance processing, and future studies in combining these approaches are planned.

4.5.4 Summary

This work suggests that (1) the brain may not perform recognition via feedforward models (2) Self-Regulatory Feedback is a major contributor to flexible processing and binding. The feedback perspective may lead to a better understanding of animals' ability to efficiently interpret and segment scenes.

Acknowledgements I would like to thank Eyal Amir, Robert Lee DeVille, Dervis Vural, Ravi Rao, and Cyrus Omar for discussion and suggestions; Cyrus Omar also assisted with simulation software for testing. This work was in part supported by the U.S. National Geospatial Agency and IC Postdoc Grant HM1582-06–BAA-0001.

References

1. Achler T (2007) Object classification with recurrent feedback neural networks. In: Evolutionary and bio-inspired computation: theory and applications. Proc SPIE, vol 6563

2. Achler T (2008) Plasticity without the synapse: a non-Hebbian account of LTP. Society for Neuroscience poster, Washington
3. Achler T (2009) Using non-oscillatory dynamics to disambiguate simultaneous patterns. In: Proceedings of the 2009 IEEE international joint conference on neural networks (IJCNN'09)
4. Achler T, Amir E (2008) Input feedback networks: classification and inference based on network structure. J Artif Gen Intell 1:15–26
5. Achler T, Amir E (2009) Neuroscience and AI share the same elegant mathematical trap. J Artif Gen Intell 2:198–199
6. Achler T, Omar C et al (2008) Shedding weights: more with less. In: Proceedings of the 2008 IEEE international joint conference on neural networks (IJCNN'08), pp 3020–3027
7. Achler T, Vural D et al (2009) Counting objects with biologically inspired regulatory-feedback networks. In: Proceedings of the 2009 IEEE international joint conference on neural networks (IJCNN'09)
8. Ackerman S. (2010) $19 billion later, Pentagon's best bomb-detector is a dog. Wired Magazine, http://www.wired.com/dangerroom/2010/10/19-billion-later-pentagon-best-bomb-detectoris-a-dog/
9. Aroniadou-Anderjaska V, Zhou FM et al (2000) Tonic and synaptically evoked presynaptic inhibition of sensory input to the rat olfactory bulb via GABA(B) heteroreceptors. J Neurophysiol 84(3):1194–1203
10. Atema J (1988) Distribution of chemical stimuli. In: Sensory biology of aquatic animals. Springer, New York, pp 29–56, xxxvi, 936 p
11. Bab-Hadiashar A, Suter D (2000) Data segmentation and model selection for computer vision: a statistical approach. Springer, New York
12. Carpenter GA, Grossberg S (1987) A massively parallel architecture for a self-organizing neural pattern-recognition machine. Comput Vis Graph Image Process 37(1):54–115
13. Chen WR, Xiong W et al (2000) Analysis of relations between NMDA receptors and GABA release at olfactory bulb reciprocal synapses. Neuron 25(3):625–633
14. Douglas RJ, Martin KA (2004) Neuronal circuits of the neocortex. Annu Rev Neurosci 27:419–451
15. Famiglietti EV Jr, Peters A (1972) The synaptic glomerulus and the intrinsic neuron in the dorsal lateral geniculate nucleus of the cat. J Comp Neurol 144(3):285–334
16. Felleman DJ, Van Essen DC (1991) Distributed hierarchical processing in the primate cerebral cortex. Cereb Cortex 1(1):1–47
17. Froemke RC, Tsay IA et al (2006) Contribution of individual spikes in burst-induced long-term synaptic modification. J Neurophysiol 95(3):1620–1629
18. Garthwaite J, Boulton CL (1995) Nitric oxide signaling in the central nervous system. Annu Rev Physiol 57:683–706
19. Herd SA, O'Reilly RC (2005) Serial visual search from a parallel model. Vis Res 45(24):2987–2992
20. Itti L, Koch C (2001) Computational modelling of visual attention. Nat Rev Neurosci 2(3):194–203
21. Koch C, Ullman S (1985) Shifts in selective visual attention: towards the underlying neural circuitry. Hum Neurobiol 4(4):219–227
22. LaBerge D (1997) Attention, awareness, and the triangular circuit. Conscious Cogn 6(2–3):149–181
23. Landisman CE, Connors BW (2007) VPM and PoM nuclei of the rat somatosensory thalamus: intrinsic neuronal properties and corticothalamic feedback. Cereb Cortex 17(12):2853–2865
24. Latecki L (1998) Discrete representation of spatial objects in computer vision. Kluwer Academic, Dordrecht
25. Marder E, Goaillard JM (2006) Variability, compensation and homeostasis in neuron and network function. Nat Rev Neurosci 7(7):563–574
26. Mcfadden FE (1995) Convergence of competitive activation models based on virtual lateral inhibition. Neural Netw 8(6):865–875
27. Mozer MC (1991) The perception of multiple objects: a connectionist approach. MIT Press, Cambridge

28. Navalpakkam V, Itti L (2007) Search goal tunes visual features optimally. Neuron 53(4):605–617
29. Pearce TC (2003) Handbook of machine olfaction: electronic nose technology. Wiley-VCH, Weinheim
30. Rachkovskij DA, Kussul EM (2001) Binding and normalization of binary sparse distributed representations by context-dependent thinning. Neural Comput 13(2):411–452
31. Rao AR, Cecchi GA et al (2008) Unsupervised segmentation with dynamical units. IEEE Trans Neural Netw 19(1):168–182
32. Rosenblatt F (1962) Principles of neurodynamics; perceptrons and the theory of brain mechanisms. Spartan Books, Washington
33. Treisman A, Gormican S (1988) Feature analysis in early vision—evidence from search asymmetries. Psychol Rev 95(1):15–48
34. Treisman AM, Gelade G (1980) A feature-integration theory of attention. Cogn Psychol 12(1):97–136
35. Turrigiano GG, Nelson SB (2004) Homeostatic plasticity in the developing nervous system. Nat Rev Neurosci 5(2):97–107
36. von der Malsburg C (1999) The what and why of binding: the modeler's perspective. Neuron 24(1):95–104, 111–125
37. Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, Amsterdam
38. Wolfe JM (2001) Asymmetries in visual search: an introduction. Percept Psychophys 63(3):381–389

Chapter 11

Artificial General Intelligence Begins with Recognition: Evaluating the Flexibility of Recognition

Tsvi Achler

Los Alamos National Labs, Los Alamos, USA

achler@gmail.com

Although many types of recognition algorithms have been developed over the past half-century, currently there is not a good way to compare the flexibility and ability to reason using recognition algorithms. Part of the difficulty is the lack of a good definition of flexibility. This chapter is dedicated to defining and evaluating flexibility in recognition. The goal is to determine flexible recognition architectures so that flexible intelligence, Artificial "General" Intelligence (AGI), can be designed on top of them.

Recognition is an essential foundation on top of which virtually every function of intelligence is based e.g.: memory, logic, internal understanding, and reasoning. Many logic and reasoning questions can be directly answered by reasoning based on recognition information e.g.: do cows have opposable thumbs? Thus AGI is inseparable from the context of recognition. It is essential to understand forms of flexible recognition structures in order to know how to store and reason based on flexible information.

The first section describes various methods to perform recognition. The second section proposes tests and metrics to evaluate flexibility, the third provides an example of applying the tests to these methods, and the forth outlines how certain recognition structures can be more beneficial for AGI.

11.1 Introduction

Many recognition algorithms have been developed over the past half century. Arguably, the most prevalent method of classification is based on learned feedforward weights $\mathbf{W}$ that solve the recognition relationship:

$$\mathbf{Y} = \mathbf{WX} \text{ or } \mathbf{Y} = f(\mathbf{W}, \mathbf{X}) \tag{11.1}$$

Vector $\mathbf{Y}$ represents the activity of a set of labeled nodes, called neurons or outputs in different literatures and individually written as $\mathbf{Y} = (Y_1, Y_2, Y_3, \ldots, Y_M)^{\mathrm{T}}$. They are considered

supervised because the nodes can be labeled for example: $Y_1 = \text{“}dog\text{”}$, $Y_2 = \text{“}cat\text{”}$, and so on. Vector $\mathbf{X}$ represents sensory nodes that sample the environment, or input space to be recognized, and are composed of individual features $\mathbf{X} = (X_1, X_2, X_3, \ldots, X_N)^{\mathrm{T}}$. The input features can be sensors that detect edges, lines, frequencies, and so on. $\mathbf{W}$ represents a matrix of weights or parameters that associates inputs and outputs.

Learning $\mathbf{W}$ weights may require error propagation and comparison of inputs and outputs, but once $\mathbf{W}$ is defined, recognition is a feedforward process. Thus the direction of information flow during recognition is feedforward: one-way from inputs to the outputs. Variations on this theme can be found within different algorithm optimizations, for example: Perceptrons (Rosenblatt, 1958), Neural Networks (NN) with nonlinearities introduced into calculation of $\mathbf{Y}$ (Rumelhart and McClelland, 1986), and Support Vector Machines (SVM) with nonlinearities introduced into the inputs through the kernel trick (Vapnik, 1995). Although these algorithms vary in specifics such as nonlinearities determining the function f, they share the commonality in that recognition involves a feedforward transformation using $\mathbf{W}$.

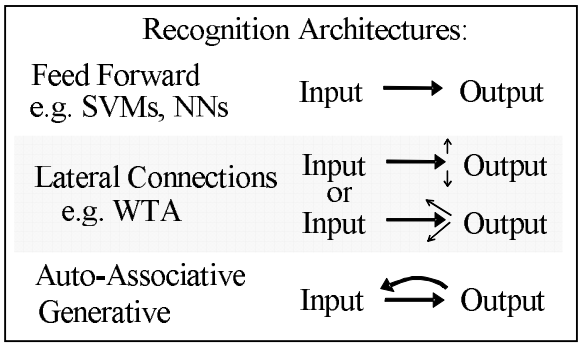

Fig. 11.1 Comparison of possible architectures used during recognition. In feedforward methods connections flow from inputs to outputs (top). Lateral connections are between outputs, or inputs that are not connected to the same node (middle). Generative or Auto-Associative connections are symmetrical and each node projects back to their own inputs (bottom).

Some recognition models implement lateral connections for competition between output nodes $\mathbf{Y}$, such as: one-vs-all, winner-take-all, all-vs-all. However even competition methods rely on initially calculating $\mathbf{Y}$ node activities based on the feedforward transformation $\mathbf{W}$.

A variation of feedforward algorithms are recurrent networks. In certain versions some of the inputs are outputs from a lower layer in the hierarchy and the network can be unfolded

into a feedforward network. This allows the processing of time. Other versions may have a limited number of outputs that project to their own inputs e.g. (Schmidhuber, 1992; Williams & Zipser 1994; Boden, 2006).

In auto-associative networks, e.g. (Hopfield, 1982), all outputs feed back to their own inputs. When part of an input pattern is given, the network completes the whole pattern.

Generative models are a variation of such auto-associative networks. The difference between generative and auto-associative networks is that the auto-associative patterns which are compared to or subtracted from the inputs. I focus on supervised generative models because they can be shown to solve the same basic recognition equations as feedforward models. Thus supervised generative and feedforward models can be directly compared.

Mathematically, generative models can be described by taking the inverse of equation (11.1):

$$\mathbf{W}^{-1}\overrightarrow{\mathbf{Y}} = \overrightarrow{\mathbf{X}} \tag{11.2}$$

Let's define $\mathbf{M}$ as the inverse or pseudoinverse of $\mathbf{W}$. The relation becomes:

$$\mathbf{M}\overrightarrow{\mathbf{Y}} - \overrightarrow{\mathbf{X}} = \mathbf{0} \tag{11.3}$$

Using this equation can be called a generative process because it reconstructs the input based on what the network has previously learned. The term $\mathbf{MY}$ is an internal prototype that best matches $\mathbf{X}$ constructed using previously learned information. The values of $\mathbf{Y}$ are the solution to the system. The fixed-points or solutions of equations (11.3) and (11.1) are identical so those same $\mathbf{Y}$'s also match the feedforward equation $\mathbf{Y} = \mathbf{WX}$.

Equation (11.3) describes the solution but does not provide a way to project input information to the outputs. Thus dynamical networks are used that converge to equation (11.3).

One method is based on Least Squares which minimizes the energy function: $\mathbf{E} = \frac{1}{2}\|\mathbf{X} - \mathbf{MY}\|^2$. Taking the derivative relative to $\mathbf{Y}$ the dynamic equation is:

$$\frac{d\overrightarrow{\mathbf{Y}}}{dt} = \mathbf{M}^{\mathrm{T}}\left(\mathbf{M}\overrightarrow{\mathbf{Y}} - \overrightarrow{\mathbf{X}}\right) \tag{11.4}$$

This equation can be iterated until $dy/dt = 0$ resulting in the fixed point solution that is equivalent to $\mathbf{Y} = \mathbf{WX}$. $\mathbf{MY}$ represents top-down feedback connections that convert $\mathbf{Y}$ to $\mathbf{X}$ domain. $\mathbf{M}^{\mathrm{T}}\mathbf{X}$ represents feedforward connections that convert $\mathbf{X}$ to $\mathbf{Y}$ domains. Both feedforward and feedback connections are determined by $\mathbf{M}$ and together emulate feedforward $\mathbf{W}$ weights.

Another way to converge to equation (11.3) is using Regulatory Feedback (RF). The equation can be written as:

$$\frac{d\overrightarrow{\mathbf{Y}}}{dt} = \overrightarrow{\mathbf{Y}}\left(\frac{1}{V}\mathbf{M}^{\mathrm{M}}\left(\frac{\overrightarrow{\mathbf{X}}}{\mathbf{M}\overrightarrow{\mathbf{Y}}}\right) - 1\right) \quad \text{where } V = \sum_{j=1}^{N} M_{ji} \tag{11.5}$$

Alternatively, using expanded notation can be written as:

$$\frac{d\overrightarrow{Y_i}}{dt} = \frac{Y_i}{\sum_{j=1}^{N} M_{ji}} \sum_{k=1}^{N} M_{ki} \left(\frac{X_k}{\sum_{h=1}^{H} M_{kh} Y_h} \right) - Y_i \tag{11.6}$$

assuming $M_{N\times H}$ dimensions for $\mathbf{M}$. Both generative-type models have the identical fixed points (Achler & Bettencourt, 2011).

Generative models, have roots in Independent Component Analysis (ICA), are commonly unsupervised and determine $\mathbf{M}$ so that sparseness is maximized e.g. (Olshausen & Field, 1996; Ziegler *et al.*, 2010). The unsupervised paradigm does not have labels, so ultimately a supervised feedforward method is still used for classification. Since the goal is to compare alternate structures that perform supervised recognition and $\mathbf{W}$ is supervised, then $\mathbf{M}$ must be supervised and sparseness is not implemented.

Some supervised generative models use a restricted Boltzmann machine which is limited to binary activation e.g. (Hinton & Salakhutdinov, 2006). Although some properties may hold throughout all types of generative models including Boltzmann machines, it is not straight forward how to represent partial recognition or equivalent range of values as $\mathbf{Y}$ in the feedforward model.

Thus less restricted supervised generative models using equations (11.5), (11.6) are evaluated e.g. (Achler, 2007; Achler, 2009). Equation (11.4) can be used as well but may require an additional parameter to converge. In-depth comparisons between supervised generative models, including differences between equations (11.4) and (11.5), are beyond the scope of this paper and will be addressed in future work.

11.2 Evaluating Flexibility

As in intelligence, evaluating robustness in recognition algorithms is not straight forward. It is also not clear what is the best metric. For example, an algorithm that is designed for a specific task and performs superbly on that task is not necessarily robust. An algorithm that shows fair performance in multiple domains may be more robust.

In order to move the field forward, particularly with the introduction of completely new methods, it is necessary to objectively evaluate and compare algorithm performance. This requires creating a benchmark. Cognitive psychologists have struggled with such metrics to evaluate human performance for almost a century and came up with IQ metrics. A similar philosophy has is suggested for AI, e.g. (Legg & Hutter, 2007).

However testing for robustness in a field such as AI poses a catch-22. Once a test is defined, it becomes a benchmark. Benchmarks commonly generate a competition for narrow algorithms that focus on the benchmark, as is the case with tests in the AI and recognition fields (e.g. Caltech, 256). Subsequently algorithms may perform well on benchmarks are not necessary the most flexible. The proposed workaround of the catch-22 is to focus on combinatorial problems combined with measurements of resources.

The proposed tests for robustness should not reward algorithms that are over-trained for the test. Thus the majority of the tests within the battery are designed to present a combinatorial explosion for a brute-force method that tries to achieve good performance only by learning specific instances of the testing suite. Multiple combinatorial tests will assure that an over-trained algorithm cannot apply to other tests. However multiple over-trained algorithms may be implemented as one algorithm to overcome the multiple tests. This is why the evaluation of resources is essential. The measurement of parameters, training, and setup costs of algorithms serves as a measure of "narrowness".

The proposed evaluation compares performance on the test set with the total number of parameters and training required. This is an Occam's razor-like metric is defined rewarding the simplest solution with the most functionality.

The test battery can be updated periodically. The overall goals of evaluation are to: 1) Design tests where performance cannot be improved by learning instances from the test suite. 2) Provide a framework where performance can be compared across algorithms. 3) Reevaluate algorithm performance and update the tests periodically to ensure flexibility and promote progress.

11.2.1 *The Testing Paradigm*

For testing the contending recognition algorithms are treated as a black box and given the same patterns. Let's define supervised input-label patterns $A \longrightarrow Z$. Thus if $\mathbf{X_A}$ is presented to the network, then it is expected to recognize it and $\mathbf{Y_A}$ should go to one. Each network to be tested is given the same input to label associations for training $(A,B,C,\ldots,Z)$. A random pattern set can be generated.

Testing is conducted by presenting patterns and pattern manipulations to the input layer $\mathbf{X}$ and evaluating based on expectations the output layer $\mathbf{Y}$ responses. The performance is pooled across tests and number of parameters included in the evaluation.

The major  f tests in the literature do not test beyond the single $\mathbf{X}$ e.g. ($\mathbf{X_{test}} = \mathbf{X_A}$, $\mathbf{X_B}$,...Ě). Thus robustness in mixtures is not explicitly evaluated in the literature.

In this work both single $\mathbf{X}$ and the sensitivity to the manipulation of mixtures of $\mathbf{X}$ are evaluated e.g. ($\mathbf{X_{test}} = \mathbf{X_A} + \mathbf{X_B}$, $\mathbf{X_A} \cup \mathbf{X_B} \ldots$).

Three types of problems are drawn upon to produce combinatorial explosions: superposition catastrophe (problems with mixtures of patterns), the binding problem (problems of grouping components of patterns), and numerosity (estimating the number patterns without individually counting each one). Unless algorithms have a powerful method of generalizing what they have learned the training required can increase combinatorially with network size.

11.2.2 *Combinatorial Difficulties of Superposition or Mixes*

A simple way to create a combinatorial difficulty is with mixtures of patterns. For example if a network can process 5 000 patterns, there are about 12 million possible two pattern combinations of those patterns, 20 billion possible three pattern combinations and so on.

In methods that over-rely on learning, the mixtures must be learned. This can quickly overcome the number of available variables. We refer to this property as a "combinatorial" explosion regardless if it is technically exponential or other function.

The mixture test evaluates algorithm performance when features of patterns are mixed or added together forming a 'Ssuperposition' (von der Malsburg, 1999; Rosenblatt, 1962).

In the superposition test, the networks are tested on patterns formed from mixtures of input vectors, for example $\mathbf{X_{mix}} = \mathbf{X_A} + \mathbf{X_B}$, $\mathbf{X_{mix}} = \mathbf{X_A} + \mathbf{X_B} + \mathbf{X_C}$ and so on. Let k represent the number of patterns mixtures superimposed in $\mathbf{X_{mix}}$. Let n represent the number of individual patterns the network knows, then the possible number of combinations increases exponentially and is given by:

$$number\_of\_combinations = \binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{11.7}$$

During testing, for each network the top k y-values are selected and their identities are compared to the patterns in $\mathbf{X_{mix}}$. If the top k nodes match the patterns that were used to compose $\mathbf{X_{mix}}$, then a correct classification for that combination is recorded. This is repeated for all possible combinations.

As the number of simultaneous patterns increases the number of possible combinations increases combinatorially. When the networks are presented with 26 patterns, and one is chosen, $k = 1$ (i.e. $\mathbf{X_{test}} = \mathbf{X_A}, \mathbf{X_B}, \mathbf{X_C} \ldots$), there are 26 possible patterns. When networks are presented with two patterns simultaneously $k = 2$, there are 325 possible combinations

of non-repeating patterns (e.g. $\mathbf{X_{mix}} = \mathbf{X_A} + \mathbf{X_B}$, $\mathbf{X_A} + \mathbf{X_C}$, $\mathbf{X_A} + \mathbf{X_D}, \ldots$). When networks are presented with three pattern combinations, $k = 3$ (e.g. $\mathbf{X_{mix}} = \mathbf{X_A} + \mathbf{X_B} + \mathbf{X_C}$, $\mathbf{X_A} + \mathbf{X_B} + \mathbf{X_D}$, $\mathbf{X_A} + \mathbf{X_B} + \mathbf{X_E} \ldots$) there are $2,600$ possible combinations. For eight simultaneous patterns (i.e. $k = 8$), there are $1,562,275$ combinations. If networks cannot generalize for super-positions, they must train for most of these combinations, i.e. most of the 1.5 million combinations. Learning $k = 8$ does not guarantee good performance with other k values. Thus potentially the networks must be trained on all k's.

In this test the superpositions are combined "noiselessly". If two patterns say $\mathbf{X_A}$ and $\mathbf{X_B}$ have the same feature X_1 and each pattern has $X_1 = 1$, then if both patterns are superimposed (pattern1 + pattern2) the value of that feature is $X_1 = 2$ in the superposition. Next we evaluate combinations with information loss.

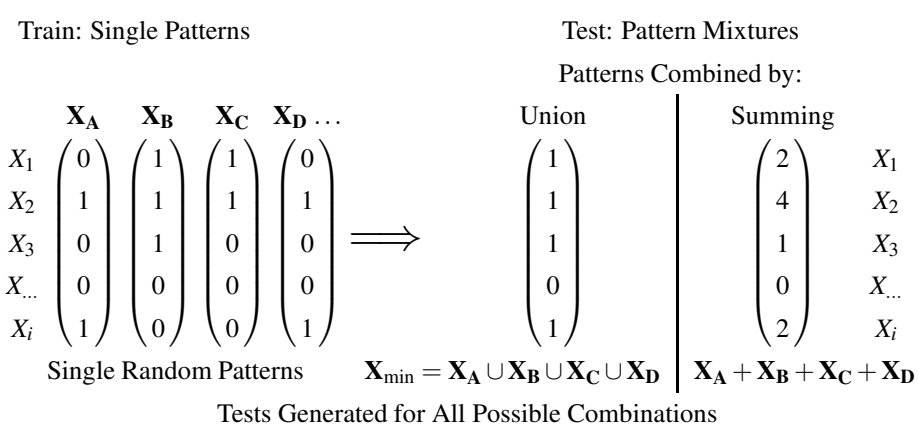

Fig. 11.2 Training with single patterns, testing with pattern mixtures. Test patterns are superpositions of single patterns combined by summing or union. Only 5 input features shown.

11.2.3 *"Occluding" Superpositions*

Flexible algorithms must function in multiple scenarios, even if fundamental assumptions of the scenarios change. For example, instead summing, overlapping features may "block" each other and their overlap may be better represented as a union or min of features.

In the union or min superposition test, the networks are tested on analogous simultaneous vectors that are combined using unions ($\mathbf{X_A} \cup \mathbf{X_B}$, $\mathbf{X_A} \cup \mathbf{X_B} \cup \mathbf{X_C}$, etc.). Not only do the same combinatorial problems in training occur with this scenario, the method for addition

may not apply to union superposition. The symbol $\cup$ will be used to represent this function and is described as a union. Note that if values within the matrixes are non-binary then the union can be represented by the minimal value at the intersection, a *min* function.

It is important to note that some information will be lost in a union. For example $\mathbf{X_{mix}} = \mathbf{X_A} \cup \mathbf{X_A}$, will be identical to $\mathbf{X_A}$, $\mathbf{X_A} \cup \mathbf{X_A} \cup \mathbf{X_A}$ etc., thus repeats cannot be decoded. However this is a useful test for the flexibility of algorithms. Since the comparison is between algorithms, any systematic information loss will affect the tested algorithms equally. Thus un-decodable aspects of test cases may reduce the number correct but will not affect overall ranking of algorithms since the algorithms are compared against each other. An example of the Union Superposition test is found in (Achler, Omar and Amir, 2008).

11.2.4 *Counting Tests*

The Superposition Catastrophe test has no more than one instance per pattern, without repeats. However, repeats (e.g. $\mathbf{X_{mix}} = \mathbf{X_A} + \mathbf{X_A}$) also provide important test cases. The ability to generalize without training specifically for the repeats is important for the ability to process or count simultaneous patterns without counting them one by one. Babies and animals have an inherent ability to estimate amounts even while demonstrating an impoverished ability to individually count or Subitz (Feigenson, Dehaene & Spelke, 2004). This ability is important to find the richest food sources and make decisions relevant to survival.

The counting test is designed to evaluate the ability to estimate amounts without individually counting. A network that can automatically process a superposition mixture such as $\mathbf{X_{mix}} = \mathbf{X_A} + \mathbf{X_K} + \mathbf{X_J} + \mathbf{X_E} + \mathbf{X_J} + \mathbf{X_K} + \mathbf{X_K} + \mathbf{X_G}$ can evaluate whether there are more K's than J's without individually counting the patterns. In the counting test, the amplitude value of y's are evaluated. For the $\mathbf{X_{mix}}$ above it is expected that Y will have $Y_A = 1$, $Y_E = 1$, $Y_G = 1$, $Y_J = 2$, $Y_K = 3$, all other Y's$= 0$.

$$number\_of\_combinations = \frac{n^k}{k!} \tag{11.8}$$

In the count test, instead of selecting the top k y-values and comparing their identity to patterns in $\mathbf{X_{mix}}$, the amplitude value of y's is compared to the number of repeats of the corresponding pattern within $\mathbf{X_{mix}}$. The number of possibilities in the count test increase even more significantly than the superposition test. An applied example of the superposition and counting test is found in (Achler, Vural & Amir, 2009).

11.2.5 *Binding Tests*

The binding problem represents another scenario with a combinatorial explosion. However this often has different meanings in neuroscience, cognitive science & philosophy. Thus we briefly review some definitions & usage.

According to "the neural binding" hypothesis, neurons within different neuronal assemblies fire in synchrony to unite different features of neuronal representations together (Gray *et al.*, 1989; von der Malsburg, 1999). This definition is part of the internal workings of a specific mechanism. It is not within the scope of this paper since algorithms are treated as black boxes.

Another definition of binding is a "unity of perception". This idea is somewhat metaphysical since is hard to objectively define and measure unity from a human's report. However there is a rich literature on errors of perception. Binding problems occur in humans when image features can be interpreted through more than one representation (Rosenblatt, 1962). An intuitive way to describe this problem is through visual illusions. For example, a local feature can support different representations based on the overall interpretation of the picture. In the old woman/young woman illusion of Fig 11.3, the young woman's cheek is the old woman's nose. Though the features are exactly the same, the interpretation is different. In humans, this figure forms an illusion because all features in the image can fit into two representations. Classifiers have similar difficulties but with simpler patterns. If a pattern can be part of two representations then the networks must determine to which it belongs. Training is used to find optimal interconnection weights for each possible scenario. However, this is not trivial for combinations of patterns and training can grow exponentially. We refine this definition using simple tests and suggest it fits in a more general mathematical framework of *set-cover*.

Fig. 11.3

The most basic binding scenario is given in figure 11.4D. Suppose a larger pattern completely overlaps with a smaller pattern: (activation of node Y_1 is determined by feature X_1, and activation of node Y_2 is determined equally by features X_1 and X_2). When presented with the larger pattern, (features X_1 & $X_2 = 1$) the representation of the smaller pattern should not predominate. The network should recognize Y_2 (settle on $Y_1 = 0$, $Y_2 = 1$). A correct classification is Y_1 when only $X_1 = 1$, but Y_2 when X_1 & $X_2 = 1$. This satisfies the set-cover problem (see below). The classifier should prefer the representation that covers the most inputs with the least amount of overlap in the inputs.

A more complex scenario occurs with the addition of Y_3 (activation of node Y_3 is determined equally by features X_2 and X_3), see figure 11.4E. Yet, the set-cover requirement holds. Since the same basic overlap exists between Y_1 and Y_2, the same interaction should remain given X_1 and X_2. However if X_1 & X_2 & $X_3 = 1$, then activation of Y_1 and Y_3 simultaneously can completely cover these inputs. Any activation of Y_2 would be redundant because X_2 would be covered twice.

For intuition nodes Y_1, Y_2, and Y_3, have been given representations of wheels, barbell, and chassis respectively.

Choosing Y_2 given X_1 & X_2 & $X_3 = 1$  is equivalent to choosing the irrelevant features for binding. If the inputs represent spatially invariant features where feature X_1 represents circles, X_3 represents the body shape and feature X_2 represents a horizontal bar. Y_1 represents wheels and thus when it is active, feature X_1 is interpreted as wheels. Y_2 represents a barbell ○—○ composed of a bar adjacent to two round weights (features X_1 and X_2). Note: even though Y_2 includes circles (feature X_1), the circles do not represent wheels (Y_1), they represent barbell weights. Thus if Y_2 is active feature X_1 is interpreted as part of the barbell. Y_3 represents a car body without wheels (features X_2 and X_3), where feature X_2 is interpreted as part of the chassis. Now given an image of a car with all features simultaneously (X_1, X_2 and X_3), choosing the barbell (Y_2) even though technically a correct representation, is equivalent to a binding error within the wrong context in light of all of the inputs.

Most classifiers if not trained otherwise are as likely to choose barbell or car chassis (Figure 11.4E). In that case the complete picture is not analyzed in terms of the best fit given all of the information present. This training is not trivial and may represent a combinatorial problem.

A solution based on set-cover automatically classifies the components and subcomponents of this problem. If X_1, X_2, $X_3 = 1$, then Y_1 and Y_3 represents the most efficient solution. If X_1, $X_2 = 1$, then Y_2 represents the most efficient solution. Set-cover also resolves basic recognition: if $X_1 = 1$, then Y_1 is the most efficient solution, if X_2, $X_3 = 1$, then Y_3 is the most efficient solution. Analysis of larger scenarios with infinite chains can be found in (Achler and Amir, 2008).

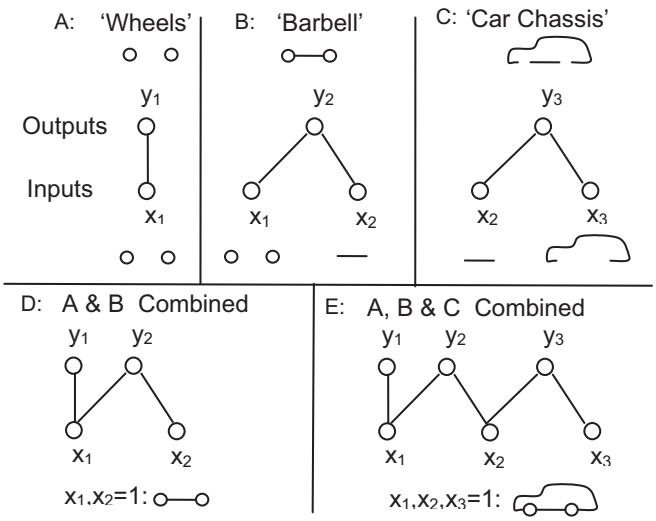

Fig. 11.4 (A–E): Modular combination of nodes displaying binding Nodes Y_1, Y_2, Y_3 (A, B & C) in combined networks (D and E). Y_1 & Y_3 represent car with wheels, Y_2 represents barbell. If X_1, $X_2 = 1$ then Y_2 should predominate (not Y_1), because it encompasses all active inputs. If X_1, X_2, $X_3 = 1$ then Y_2 should not predominate because interpreting a barbell within the car is a binding error.

11.2.6 *Binding and The Set-Cover Problem*

The notion of set cover can guide the design more complex binding tests. Given a universe of possible X's of input patterns and Y's that cover X's. Set cover asks what is the sub-set of Y's that cover any arbitrary set of X but which use the fewest Y's.

Set cover can explain the old-young woman illusion, where the most encompassing representations mutually predominate. In that illusion there are two equal encompassing representations. Recognition settles on either the old woman or young woman, and no other combinations of features. All features are interpreted as part of either representation even if the interpretation of the individual feature can vary drastically (e.g. cheek vs. nose).

Set cover is not a trivial problem to compute. The evaluation of set covering is NP-complete, but the optimization of set cover is NP-hard. This means that training networks to account for every possible cover is not practical because it may require every possibility to be trained, exposing another combinatorial explosion.

Thus networks that can "look beyond what is learned" should be able to resolve set cover. Simple examples that require set cover can be introduced to a test set. Other examples of binding tests can be found in (Achler & Amir, 2008) and (Achler, 2009).

11.2.7 *Noise Tests*

Random noise is often used as a metric for flexibility. Random noise can be predicted and trained. Since training with noise does not necessarily result in a combinatorial explosion, this is less favored. However resistance to random noise is important, and commonly used measure, it is added to the battery. Noisy stimuli can be generated using real-valued random exponential noise with mean amplitude μ, added to the prototypical stimulus. Noisy test vectors can be given to the classifiers and their result is compared to its original labels. The percent of patterns correctly identified for that noise level can be recorded. Other systematic non-random noise scenarios can be tested as well.

11.2.8 *Scoring the Tests*

The *Intelligence Quotient* (IQ) scoring system is borrowed from the field of psychology because it provides a familiar reference. Here each algorithm tested is like an individual. Performance on the battery of tests is pooled and the algorithms are ranked relative to the average of all algorithms. Performance is ranked on the Gaussian bell curve with a center value (average IQ) of 100, each standard deviation is 15 points. IQ value will be 100 for the average performing individuals (> 100 for better than average, < 100 for less than average), regardless of the actual score on a particular sub-test.

The absolute value of performance an algorithm achieved in a single test is not relevant since comparisons are the ultimate evaluation. For example, suppose one algorithm's performance on a particular sub-test in the battery is low, say 10% correct. Since IQ values only reflect differences from average performance, if the average is 7% correct, that performance provides a better-than-average contribution to that algorithms IQ score for that test. Tests can be simply combined to give an aggregate for the battery. For example:

$$\text{Battery\_IQ} = \text{Average}(\text{IQ\_numerosity} + \text{IQ\_superposition} + \text{IQ\_binding} + \text{IQ\_noisy} + \cdots) \tag{11.9}$$

The IQ ranking system, evaluates algorithm flexibility while measuring and encouraging performance improvement. Additional tests are easy to include into the battery and scoring system.

11.2.9 *Evaluating Algorithms' Resources*

One advantage of AI testing over humans testing is the ability to evaluate the number of training trials and resources used by the algorithm. An algorithm with many parameters

and extensive training may do slightly better than another algorithm that was not given as much training and does not have as many free parameters. However, a slightly better performance at the cost of more training and variables may actually be less flexible. The algorithm requiring the least amount of training, with the least amount of free parameters, while performing well would be the most desirable.

Thus we have included a measure of resources to the Test score that accounts for the number of training epochs and free variables. These factors would modify the final test score to give a final AI_IQ score.

$$\text{AI\_IQ} = \frac{\text{abilities}}{\text{resources}} = \frac{\text{Battery\_Score}}{\text{training\_epochs} + \text{variables} + \text{training}} \tag{11.10}$$

It is the philosophy of evaluation – rather than the details – that are important for flexibility evaluation. Flexible AI should require less degrees of freedom and apply to a greater number of scenarios (without extensive retraining for each scenario). The purpose of the metrics is to encourage such capabilities, limit degrees of freedom but maximize applicability.

11.3 Evaluation of Flexibility

This section provides concrete examples testing that compare the flexibility of feedforward methods to generative and other methods. This analysis is not complete but provides in-depth examples. We begin by defining random patterns outlined in Section **??**: using 26 patterns and 512 input features. SVM, a Winner-take-all (WTA) algorithm, generative RF, and three different versions of NN algorithms are compared. Naïve NN's were trained only on single patterns, NNs were trained on single and 2 pattern mixes, and NNs were trained with WEKA.

The purpose of the publicly available *Waikato Environment for Knowledge Analysis* (WEKA) package (Witten & Frank, 2005) is to facilitate algorithm comparison and maintains up to date algorithms.

We begin by training a NN. The NNs are trained via backpropagation with momentum. $100,000$ training examples are randomly chosen with replacement from the prototypes. The number of hidden units used is increased until it was sufficient to learn the input-output mapping to a component-wise tolerance of 0.05. Resulting NNs have on average 12 hidden units. The learning rate was .1, the momentum was 1 and there is a bias unit projecting to both hidden and output layers with value of 1.

Let's evaluate the number of parameters for NN: 26 output nodes, 512 input nodes, 12 hidden units, bias unit, momentum, component-wise tolerance, and learning rate. There

are 551 variables so far. However during training NNs were created with 1-11 hidden units which did not perform well. There were also $100,000$ training episodes for NN. All of those resources should be included in the evaluation because that is more than $1,200,000$ episodes of training.

After this, suppose performance on the mixtures is poor and to improve performance the network is trained on 2-pattern mixtures. That represents 325 more training patterns and more epochs. Those should be included as well.

There are 26 training episodes for RF. No training was done for combinations.

The Winner-take-all algorithm is trained the same way as the RF algorithm however each output node has inhibitory connections to all other nodes. Since all of the output nodes are uniformly inhibitory, only one variable is added. Like the NN WEKA algorithm the SVM algorithm was simulated as a black box extension of WEKA. SVMs do not have hidden units so the number output and input nodes should be the same as RF. However SVMs can have kernels and those may be governed by many variables. Since WEKA is treated as a black box these are not counted and the number of training episodes is not counted as well.

The WTA algorithm is evaluated for multiple patterns by 1) finding the most active node, 2) determining its identity, 3) inhibiting it, 4) finding the next most active node.

11.3.1 *Superposition Tests with Information Loss*

Superposition pattern mixtures were combined using a union (or max function). Some information is lost in the mixtures. Following (Achler, Omar, Amir, 2008) let's look at test performance. Two more networks that were not present in the original paper are included here: SVM and winner-take-all WTA. Algorithm names are marked in bold.

RF performed well within all mixtures showing more flexibility. The number of parameters in WEKA are grossly underestimated, because it is being treated as a black box here. A correct number of parameters is critical to evaluating flexibility.

11.3.2 *Superpositions without loss*

Next superposition tests where pattern mixtures are combined using an addition function, without information loss, are evaluated following (Achler, 2009).

Comparing this test to the occluding test, winner-take-all performed better while **NN** and **SVM** performed worse.

Table 11.1 (a) "Occluding" Superposition: evaluation of performance given mixtures with information loss. Raw scores for: 26 single patterns (left), 2-patern mixtures and 4-pattern mixtures (right).

number mixed:	**k = 1**	**k = 2**	**k = 4**
combinations:	**26**	**325**	**14 950**
RF	100%	100%	90%
NN naïve	100%	52%	4%
NN 2-pat	100%	92%	32%
NN WEKA	100%	91%	28%
SVM WEKA	100%	91%	42%
WTA	100%	85%	24%

Table 11.1 (b) Total score for the 15301 occluding mixtures and analysis. Raw overall score (top row), followed by IQ values based on those scores, then the number of parameters and training episodes for each algorithm. * indicates estimated number of parameters. AI_IQ scores (bottom) are sensitive to the number of parameters.

	RF	**NN naïve**	**NN 2-pat**	**NN WEKA**	**SVM WEKA**	**WTA**
Score (%)	90.2	5.2	33.4	29.5	43.1	25.4
IQ	128	83	98	96	103	93
Parameters	565	100 551	100 551	565*	565*	566
AI_IQ	126	86	86	99	105	97

Table 11.2 (a) Performance for superposition mixtures without information loss. 26 single patterns (left), 2-patern mixtures and 4-pattern mixtures (right) were tested. RF correctly recognized all patterns in all 325 $k = 2$ and 14950 $k = 4$ combinations.

number mixed:	**k = 1**	**k = 2**	**k = 4**
combinations:	**26**	**325**	**14950**
RF	100%	100%	100%
NN WEKA	100%	91%	4%
SVM WEKA	100%	94%	8%
WTA	100%	91%	42%

11.3.3 *Counting Tests*

Next let's evaluate counting tests where repeating patterns are combined in the mixtures using an addition function. Following (Achler, Vural & Amir, 2009), however only regulatory feedback is tested. Unfortunately WEKA could not test more than 15 000 tests. It appears not designed for such large number of tests. This is an indication that the literature

Table 11.2 (b) Total score for the 15301 superposition mixtures (top row), followed by IQ values based on those scores, then the number of parameters and training episodes for each algorithm. * indicates estimated number of parameters. AI_IQ scores (bottom) are sensitive to the number of parameters.

	RF	NN WEKA	SVM WEKA	WTA
Score (%)	100	6	10	43.1
IQ	121	88	90	101
parameters	565	565*	565*	566
AI_IQ	131	89	90	105

is not focusing enough on manipulations during recognition: only testing single patterns: e.g. $\mathbf{X_A}$, $\mathbf{X_B}$ and so on. It is hoped that future versions will allow more testing. Winner-take-all by definition is not well suited for repeats because there is no mechanism to control selection of a node several times.

The tests are based on the same 26 patterns used in the superposition. 17,576 possible combinations of 3 and 456,976 possible combinations of 4 pattern repeats were tested. **RF** scored 100% on all tests. Although this is an important part of the test battery, this test is no included in the overall score because of the limited comparisons are available through WEKA.

11.3.4 *Binding Scenarios*

The simplest and next simplest binding scenarios require retraining of networks. In the simplest binding scenario there are two output nodes and two input nodes. Networks trained on the simplest scenario with two nodes perform correctly. Thus **NN WEKA**, **SVM WEKA**, **Winner-take-all**, **RF** all score 100%. Again, the number of training epochs, hidden nodes and kernels parameters in WEKA are not evaluated since WEKA is considered a black box. Thus the performance of all methods are considered approximately equal.

In the next simplest binding scenario there are three inputs and three output nodes. Not algorithms perform this correctly. In the case of all inputs active, **RF** decides on the correct two patterns that best fit the inputs. **NN WEKA** and **SVM WEKA** settle 50% on two representations that capture most of the inputs but do not cover all of the inputs in an efficient manner. Other cases of input patterns were correctly matched.

Of the 4 possible of input patterns, **RF** achieved 4/4, **SVM & NN WEKA** and **Winner-take-all** achieved 3/4. Thus scores for this test are 100%, 75%, 75%, 75% respectively.

This is a simple demonstration of binding. Further tests are envisioned to be developed for more complex binding/set-cover scenarios.

11.3.5 *Noise Tests*

Tests were performed on the 26 patterns with random exponential noise with mean amplitude μ, added to the prototypical stimulus. As part of tests NN were trained on patterns with $\mu = 0.25$ and $\mu = 0.15$, **NN $\mu = 0.25$** and **NN $\mu = 0.15$** respectively. Let's look at test performance (from Achler, Omar, Amir, 2008).

Table 11.3 (a) Comparison of performance in given random noise. Trained NNs on noise improves performance. Naïve RF performs better than naïve NN, but NN WEKA performed best.

	Input Noise Level μ						
	0.1	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**
RF	100%	100%	95%	85%	68%	47%	29%
NN naïve	100%	78%	43%	24%	12%	9%	8%
NN $\mu = 0.15$	100%	100%	100%	81%	53%	28%	17%
NN $\mu = 0.25$	100%	100%	100%	97%	79%	49%	28%
NN WEKA	100%	100%	100%	100%	98%	91%	76%

Table 11.3 (b) Total score for the 182 tests (top row), followed by IQ values based on those scores, then the number of parameters and training episodes for each algorithm. * indicates estimated number of parameters. AI_IQ scores (bottom).

	RF	**NN naïve**	**NN $\mu = 0.15$**	**NN $\mu = 0.25$**	**NN WEKA**
Score (%)	75	39	68	79	95
IQ	103	76	98	106	117
parameters	565	100 551	100 551	100 551	565*
AI_IQ	120	86	86	86	129

The number of variables and training epochs for NN are the same as Section 11.3.1 with 100 551 total. Again we do not apply variables and training for WEKA at this point. All NNs that were well-trained for noise performed better than **RF**. However **RF** was more noise resistant without training than **NN naïve**.

11.3.6 *Scoring Tests Together*

The proposed test suite includes: superposition, occluding superposition, counting, binding, and noise. However at this point the most complete results available compare RF and NN WEKA in: superposition, occluding superposition, binding, and noise. The total score weighing all weighted tests equally is RF 91.3; NN WEKA 51.4*. IQ RF 111; NN WEKA 89. Again, these numbers do not properly account for the number of parameters and training in WEKA. Ideally the all resources should be included including hidden layer variables and WEKA training. However the goal here is to indicate how flexibility can be evaluated and quantified. From this perspective this limited demonstration is sufficient. This score and demonstration shows how flexibility can be evaluated across a set of tests with combinatorial difficulties.

11.3.7 *Conclusion from Tests*

The test battery demonstrates fundamental differences in flexibility between M (RF) and W (NN). Despite the black box designation for WEKA, RF still performed better than NN and SVM. Recognition based on M can be quantified as more flexible. This is demonstrated whether the mixtures are summed together (Achler, 2009) in intensity or combined as a union (Achler, Omar, Amir, 2008). The mixtures can contain multiple additions of the same pattern (repeats) and the networks are able to determine the pattern mixture's numerosity (Achler, Vural, Amir, 2009). They are also able to resolve certain binding scenarios: i.e. to determine whether subcomponent parts belong together (Achler & Amir, 2008).

Comparing performance between generative models, using the least-squares method, equation (11.4) we obtain the same results except for the union cases and some instances of the binding cases. The instances of binding and unions that did not settle on the same solutions had input patterns outside the fixed points or linear combinations of the fixed points. In those cases both models converged, however differing results can be expected in different methods solving the generative model as no guarantees are made outside of the fixed points. The significance of the differences between methods is beyond the scope of this paper and will be discussed in future work.

11.4 From Recognition to AGI

Lastly this section outlines how a recognition system based on **M** is better suited to answer logic perception questions such as "do cows have opposing thumbs?" Suppose that a recognition system has output nodes associated with "cow" and "limbs" and "thumb". Because **M** is bidirectional biasing an output node generates an image. When an output is biased, its activity is artificially increased (see Achler & Amir, 2008). This generates an image through the top-down component of **M**. The image can be used to activate inputs as if an image was given to the system. This is an imagination or analysis mode. From the "mental image" reconstruction, recognition of other outputs can be measured. For example the image of thumb can be evaluated when "cow" is biased. This is a method to generalize perception into AGI and to be able to do mental image manipulations. Moreover this model does not require every node to be connected to every other one. This means output nodes for "thumb", "cow", and "limbs" do not need to be directly connected to each other. This avoids combinatorial problems both from the perspective of connections and parameters because all outputs do not need to be directly connected to each other.

In contrast, recognition models based on **W** encompass a feedforward structure. There is no convenient way to transverse from an output node back to the expected inputs. This limits manipulative ability and subsequently AGI possibilities. Further analysis and development of this system is described in future chapters.

In summary **M** is a better way of performing recognition because it is more flexible. Moreover, its bidirectional properties have benefits for developing AGI reasoning.

Acknowledgments

A part of this work was supported by the Notational Geospatial Agency and Los Alamos National Labs. I would like to thank Luis Bettencourt, Garrett Kenyon, Peter Loxley, Tanya Berger-Wolfe and those who helped review this work for valuable comments.

Bibliography

[1] Achler T. (2009). Using Non-Oscillatory Dynamics to Disambiguate Simultaneous Patterns. IEEE IJCNN 2009: 3570.

[2] Achler T. Amir E. (2008). Input Feedback Networks: Classification and Inference Based on Network Structure. Artificial General Intelligence 1: 15–26.

[3] Achler T., Amir E., (2008). Hybrid Classification and Symbolic-Like Manipulation Using Self-Regulatory Feedback Networks, Proc 4'th Int Neural-Symbolic Learning & Reasoning Workshop.
[4] Achler T., Bettencourt L., (2011). Evaluating the Contribution of Top-Down Feedback and Post-Learning Reconstruction, Biologically Inspired Cognitive Architectures AAAI Proceedings.
[5] Achler T., Omar C., Amir E. (2008). Shedding Weights: More With Less. IJCNN 2008: 3020–3027.
[6] Achler T. Vural D., Amir E. (2009). Counting Objects with Biologically Inspired Regulatory-Feedback Networks. IEEE IJCNN 2009: 36–40.
[7] Boden M. (2006). A guide to recurrent neural networks and backpropagation. <http://www.itee.uq.edu.au/~mikael/papers/rn_dallas.pdf>.
[8] Feigenson L., Dehaene S., & Spelke E., (2004). Core systems of number. Trends Cogn. Sci. 8 (7): 307–14.
[9] Gray C.M., Konig P., Engel A.K., Singer W. (1989). "Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties." Nature 338 (6213): 334–7.
[10] Hinton G.E., & Salakhutdinov R.R. (2006). Reducing the dimensionality of data with neural networks. Science, 313 (5786), 504–507
[11] Hopfield J.J. (1982) Neural networks and physical systems with emergent collective computational abilities, PNAS, vol. 79 no. 8 pp. 2554–2558.
[12] Hyvärinen A., Hurri J., Hoyer P.O. (2009). Natural Image Statistics, Springer-Verlag.
[13] Legg S. and Hutter M. Universal intelligence: A definition of machine intelligence. Minds & Machines, 17 (4):391–444, 2007.
[14] Olshausen B.A., Field D.J., (1997) Sparse coding with an overcomplete basis set: A strategy employed by V1? Vision Research 37:3311–3325.
[15] Rosenblatt F. (1962). Principles of neurodynamics; perceptrons and the theory of brain mechanisms. Washington, Spartan Books.
[16] Rumelhart D.E., & McClelland J.L. (1986). Parallel distributed processing: explorations in the microstructure of cognition. Cambridge, Mass.: MIT Press.
[17] Schmidhuber J. (1992). Learning complex, extended sequences using the principle of history compression. Neural Computation, 4 (2):234–242.
[18] Vapnik V.N. (1995). The nature of statistical learning theory. New York: Springer.
[19] von der Malsburg C. (1999). The what and why of binding: the modeler's perspective. Neuron, 24 (1), 95-104, 111–125
[20] Williams R.J., Zipser D. (1994). Gradient-based learning algorithms for recurrent networks and their computational complexity. In Back-propagation: Theory, Architectures & Applications. Hillsdale, NJ: Erlbaum.
[21] Witten I.H., & Frank E. (2005). Data mining: practical machine learning tools and techniques (2nd ed.). Amsterdam; Boston, MA: Morgan Kaufman.
[22] Zeiler M.D., Kirshnan D., Taylor G.W., Fergus R. (2010). Deconvolutional Networks, Computer Vision & Pattern Recognition CVPR.

Shedding Weights: More With Less

Tsvi Achler, Cyrus Omar, and Eyal Amir

Abstract—**Traditional connectionist models place an emphasis on learned weights. Based on neurobiological evidence, a new approach is developed and experimentally shown to be more robust for disambiguating novel combinations of stimuli. It does not require variable weights and avoids many training related issues. This approach is compared with traditional weight-learning methods. The network is better able to function in different scenarios and can recognize multiple stimuli even if it is only trained on single stimuli.**

I. Introduction

QUALITATIVELY intelligence can be described ability to apply learned information to a new scenario. Classification is an important aspect of this intelligence. We review two connectionist classifiers that can be considered biologically plausible and evaluate their ability to generalize. These classifiers are 1) *Neural Networks* (NNs) and 2) a new type of network we call *Regulatory Feedback Networks* (RFNs). We argue that the most 'intelligent' connectionist networks require the least amount of computational resources, variables and connections, while performing successfully in multiple unforeseen scenarios.

Our biologically motivated network is inspired by the massive number of feedback connections found in the brain [1-3]. Due to its feedback structure it is called *Regulatory Feedback Networks*. Compared to traditional NNs, RFNs perform simpler processing during the training phase, but more extensive processing during testing phase. Instead of determining connection weights based on the training set like NNs, only positive binary associations are determined (e.g. *feature1* implies *X*). Negative associations (e.g. *feature1* implies not-*Y*) are not determined, but inferred during the test phase through feedback connections. This allows the network to be more robust and function outside of its training distribution.

We outline the biological evidence for RFNs and compare them with traditional NNs. The comparisons explore 1) the type of problems each addresses 2) the training requirements 3) resources needed and 4) the number of variables required. A similar analysis can be extended to other common methods such as *Support Vector Machines* (SVMs) or nonparametric computational methods. However, the focus of this paper is limited to biologically plausible connectionist models.

Our stimuli are designed to bring the key issues into the forefront that may be lost in large implementations. The scenarios include: 1) ideal stimuli degraded by random noise 2) multiple stimuli simultaneously, such as may occur with a poorly segmented scene. RFNs show particular promise in detecting novel combinations of multiple stimuli, even though RFNs require: less training, less variables and avoid many training related questions.

Our results are significant because they illustrate the problems associated with connection weights and how a regulatory feedback approach can resolve some of them.

Manuscript received December 15, 2007. This work was supported in part by the U.S. National Geospatial Agency Grant HM1582-06--BAA-0001.

T. Achler is with the Computer Science Department, University of Illinois at Urbana Champaign, Urbana, IL 61801 USA (217-244-7118; fax: 217-265-6591; e-mail: achler@ uiuc.edu).

E. Amir is with the Computer Science Department, University of Illinois at Urbana Champaign, Urbana, IL 61801 USA (e-mail: eyal@cs.uiuc.edu).

II. Background

A. Connection weights

The idea of adjusting connection weights has been a cornerstone throughout the development of connectionist models [4-6]. The advantage of this method is that in combination with learning algorithms it can have a good degree of autonomy.

However, it requires high degrees of freedom where numerous sets of weights can be chosen for virtually any problem. Weights are adjusted per task for specific applications. Without appropriately selected training sets weight learning can suffer from difficulties such as over-fitting, local minima and catastrophic interference; 'forgetting' of previously learned tasks [7,8]. Furthermore, the function of each unit in relation to its structure can be unclear.

These learning algorithms are difficult to describe in terms of biologically viable neural mechanisms. Most learning theories, such as backpropagation, employ feedback connections only during learning [6]. The feedback connections are not used during classification.

Lastly, recent studies of neuron processing, challenge the idea that synaptic properties can be described by trained connection weights. For example, studies of homeostatic plasticity indicate that synapse strengths appear to maintain a fixed amount of total network activity [9,10].

B. Limits of Learning and Test Distribution Assumptions

Connectionist networks such as NNs are trained with the assumption that the training distribution is similar to the testing distribution [11]. This limitation allows the correlation between input features to outcomes to be determined a-priori through a training set. Unfortunately, this limitation is commonly violated in the natural environment [12] such as in a scene with many stimuli. Suppose a network is trained on stimulus *A* and *B* presented by themselves. If they appear side-by-side their

simultaneous appearance is outside the training distribution. This distribution limit, may limit the network's 'intelligence'.

With efforts to carefully train and avoid of over-training, NNs can maintain some ability to generalize [13]. However, this does not resolve the problems. For example, NNs can be trained to recognize A & B simultaneously. However, if C is separately trained, then simultaneous appearance of both A & C, or B & C still fall outside of the training distribution. Training NNs for every pair of possible stimuli (or triplets, quadruplets, etc.) may require a huge amount of training to cover unforeseen combinations. This is combinatorialy impractical.

To overcome these limits, a scene is often segmented into smaller individual stimuli. But, determining segmentation boundaries is not trivial [14]. Even if feedback or attention processes are employed to guide segmentation [15, 16], segmentation is not flawless. Furthermore, humans can resolve certain stimuli without segmentation [17]. Thus, combined over-reliance on segmentation and distribution limits may result in suboptimal performance.

C. *Importance of Feedback in the Brain*

The ability to modify information in earlier pathways appears to be an important component of recognition processing. Feedback connections can be found in virtually all areas of brain processing. The thalamic system (including thalamic nuclei the Lateral Geniculate Nucleus-LGN, and Medial Geniculate Nucleus-MGN) projects to the cerebral cortex and receives extensive cortical projections back. Relay cells feed forward to the cortex and pyramidal cells feed back to the cortex. These connections have been described as forming a ubiquitous 'triad' structure of regulatory feedback [18,19]. Furthermore, the thalamus may receive as many connections from the cortex as it projects to the cortex [19].

The *Olfactory Bulb* (OB), which processes odors, is analogous to the thalamus. The OB is a separate, modular structure which can easily be studied. Compared to visual or auditory signals, odorous signals are generally reduced in spatial fidelity [20]. Thus odorant processing involves primarily recognition processing as opposed to visual or auditory processing which can encompass both recognition and localization.

Regulatory feedback modulation of early processing appears not one, but two levels of processing within the OB: the local and global circuits. The local circuit, found within the OB glomeruli sub-structure, receives inputs from the olfactory nerve, and connects to mitral/tufted output cells. The output cells simultaneously activate juxtoglomerular cells (Q cell equivalent) which pre-synaptically inhibit the olfactory nerve axons. The global circuit, found between the OB and olfactory cortex, receives inputs through the mitral/tufted cells, and projects information back to granule cells within the OB. These granule cells (another Q cell equivalent) inhibit the mitral/tufted cells.

This feedback establishes nonlinear mechanisms to alter cell activation, based on its previous activity. Such mechanisms can be found within dendritic-dendritic connections involved in coincidence detection (via intercellular NMDA channels). These channels are found in both granule cells [21] and juxtoglomerular cells [22]. Together GABA channels (inhibitory connections), calcium & NMDA channels (multiplicative dynamics) and feedback form an regulatory feedback system of inhibition [23,24]. This forms the basis of RFNs.

Thus, 'intelligent' classifiers that can generalize outside the training distribution can be more biologically plausible, require simpler training and tolerate segmentation errors.

III. Regulatory Feedback Networks

In light of this evidence, RFNs use top-down regulatory feedback to modify input activation. The modified input activity is then re-distributed to the network. This is repeated iteratively to dynamically determine stimuli relevance. In this manner top-down regulatory feedback determines the relevance of inputs to an output node.

This model does not assume calculations are based on predetermined connection weights. All input features are connected equally to associated output nodes. Thus each representation is equally connected to all of its parts. Since connection weights are not relevant, only qualitative relations between feature membership to classes need to be determined during setup; e.g. $y_1 \epsilon \{x_1, x_3 \ldots\}$, $y_2 \epsilon \{x_2, x_3 \ldots\}$. Only positive associations between inputs and outputs are encoded. A potentially large number of negative associations are not encoded. Though RFNs are nonlinear, they are easy to simulate. They are well behaved: stable and maximally bounded by the values of the inputs.

A. *Model Function*

RFN can be qualitatively said to inhibit *ambiguous* inputs in order to amplify the discriminatory ability of the classifier. An input can be said to be *ambiguous during classification* if the current candidate representations of the input tend to share it. This distinction is crucial – traditional NN notions of a feature being *uninformative* are associated statically between representation and its input feature(s). Thus NN feature extraction methods will assign reduced weights to features that are generally uninformative as determined by the training set. Using our approach features weights are not assigned. Subsequently, a feature's information content can be dynamically evaluated under different contexts. This is robust because it does not require the distribution of noise to be known in advance or assumed in the training set. Thus, RFNs are more flexible towards unpredicted combinations of priors since no weighted relations between features or nodes are defined a-priori.

B. *Model Schematic*

RFN is unique due to the tight association between input features and outputs representations. This is implemented by a triad of interconnections between an input, the output it supports and feedback from that output (Figure 1). Every input has a corresponding feedback 'Q', which samples the output processes that the input cell activates. The feedback modulates the input.

Every output must project to the feedback Q processes that correspond to its inputs. For example if an output process receives inputs from I_1 and I_2 it must project to Q_1 and Q_2. If it receives inputs from I_1, it only needs to project to Q_1.

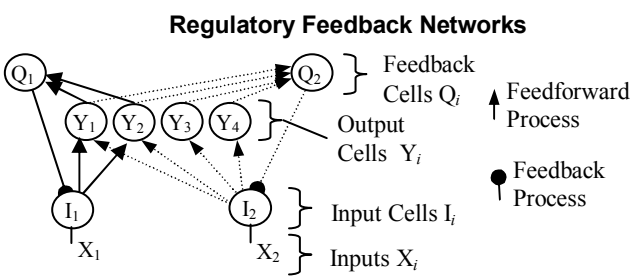

Figure 1: Regulatory Feedback Schematic. Every feed-forward connection has an associated feedback connection. If I_1 projects to Y_1 & Y_2, then Q_1 must receive projections from Y_1 & Y_2 and feedback to the input cell I_1. Similarly if I_2 projects to Y_1, Y_2, Y_3, & Y_4, then Q_2 receives projections from Y_1, Y_2, Y_3, & Y_4 and projects to I_2.

This creates a situation where an output cell can only receive full activation from an input if that input's Q is low. The Q is low if the sum of activation of the outputs that use that input is low. Thus, if representations that share the input are very active, no cell will receive full activation from that input. If outputs share inputs, they inhibit each other at their common inputs, forcing the outcome of competition to rely on other non-overlapping inputs. The more cells have overlapping inputs, the more competition exists between them. The less overlap between two output cells, the less competition, more independent from each other the output cells can be.

The networks dynamically test recognition of representations using 1) regulatory feedback to the individual inputs of representations 2) modifying the next input state based on the input's use 3) re-evaluating representations based on new activity. Steps 1-3 are continuously cycled through. This requires a tight association between inputs and outputs and feedback processes.

RFN is flexible because it doesn't a-priori define which input is ambiguous. Which input is ambiguous depends on which representation(s) are active which in turn depends on which stimuli and task are being evaluated.

C. Equations

This section introduces general nonlinear equations governing RFN. Borrowing nomenclature from engineering control theory, this type of inhibitory feedback is negative feedback, in other words stabilizing or regulatory feedback.

For any output cell Y denoted by index a, let N_a denote the input connections to cell Y_a. For any input cell I denoted by index b, let M_b denote the feedback connections to input cell I_b. The feedback to input I_b is defined as Q_b. Q_b, is a function of the sum of activity from all cells Y_j that receive activation from that input:

$$Q_b = \sum_{j \in M_b} Y_j(t) \qquad (1)$$

Input I_b is regulated based on Q_b, which is determined by the activity of all the cells that project to the input, and driven by X_b which is the raw input value.

$$I_b = \frac{X_b}{Q_b} \qquad (2)$$

The activity of Y_a is a product of its previous activity and the input cells that project to it. This property can arise biologically from NMDA channels that are found within neuron membranes. These channels are mediated by previous neuron activity. If a self-multiplicative (delay) term is not included in *equation 3*, the network can immediately change values and the feedback will not be a function of previous activity. Thus the equations are designed so the output cells are proportional to their input activity, inversely proportional to their Q feedback and also depend on their previous activity [1-3]. A simple division was chosen for *equation 2*, because a subtraction does not guarantee that the stabilizing feedback will be proportional in size to a perturbation. n_a represents the number of processes in set N_a.

$$Y_a(t+\Delta t) = \frac{Y_a(t)}{n_a} \sum_{i \in N_a} I_i \qquad (3)$$
$$= \frac{Y_a(t)}{n_a} \sum_{i \in N_a} \frac{X_i}{Q_i} = \frac{Y_a(t)}{n_a} \sum_{i \in N_a} \left(\frac{X_i}{\sum_{j \in M_i} Y_j(t)} \right)$$

D. Stability

Stability of these equations and other variants have been previously analyzed [25]. In RFN all variables are limited to positive values. Thus the values of Y can not become negative and have a lower bound of *0*. Furthermore the upper values of Y are bounded as well. By definition every input (in N_a) to which Y_a projects, will have Y_a in the feedback branches (M_b) of input I_b. To achieve the largest possible value, all other cells should go to zero. The equation then reduces to:

$$Y_a(t+\Delta t) \le \frac{1}{n_a} \sum_{i \in N_a} \left(\frac{Y_a(t) \cdot X_{\max}}{Y_a(t)} \right) = \frac{1}{n_a} \sum_{i \in N_a} X_{\max} \le \frac{X_{\max} \cdot n_a}{n_a} = X_{\max}$$

If X values are bounded by $X_{max}=1$, then each Y_a is bounded by *1*. Thus the values of Y are bounded between by the amplitude of the inputs and *0*. The picard existence and uniqueness theorem states that if a nonlinear differential equation is bounded and is well behaved locally then it will have a unique solution, i.e.[26].

E. Learning Requirements & Combinatorial Plausibility

For RFN setup, input features active in a labeled training example are connected in a positive binary fashion to their representative output node. More sophisticated methods can include clustering and pruning [27], but are beyond the scope of this paper. Learning is simpler in RFNs because no connection weights are involved. The number of connections required per cell is a function of the number of inputs the cell uses. Addition of a new cell to the network requires only that it forms symmetrical connections to its inputs and not directly connect with the other output cells.

Consequently, the number of connections of a cell representation is independent of the size or composition of the classification network, allowing large and complex feedback networks to be combinatorially and biologically practical [3].

F. Homeostatic Plasticity

Lastly, RFNs to some extent incorporate Homeostatic Plasticity. Homeostatic Plasticity describes the phenomena where biological neurons have been shown to auto-regulate baseline activity by normalizing synapse strength across all synapses through multiple cellular mechanisms [9,10]. In the regulatory feedback network each classifier cell strives for a total activation of 1. If a cell has N inputs each connection contributes 1/N activity. This guarantees that the cell will be as active as its most active inputs. However network function is not strictly dependent on this homeostatic criterion.

With these seemingly simple connections the RFN model can behave informatively in complex scenarios.

IV. Experiment and Network Set-up

This section describes the setup of the networks and methods of the experiments. RFNs and NNs are evaluated using letter examples. The stimuli are 26 letters of the alphabet each composed of 5x5 pixels. Experiments test the networks on noisy and simultaneous presentations of the letters. A feature extractor is used to allow several letters to be represented at once in the visual field.

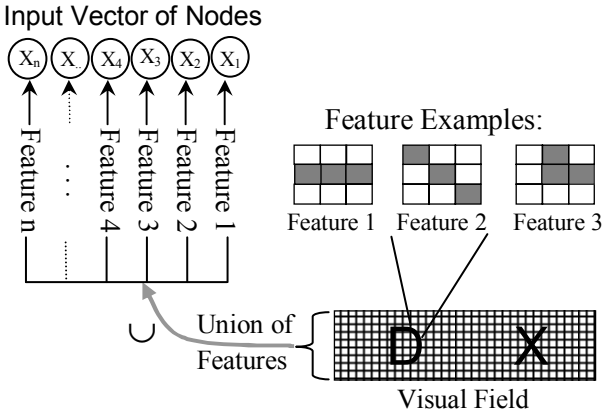

Figure 2: Feature Extractor. If a feature pattern is present anywhere in the visual field, its feature node is activated. The feature nodes serve as an input vector to the classifiers.

The extractor's features are embodied by a 3x3 pixel grid where each possible pattern is considered a feature. Thus there are 512 possible features. The 3x3 pixel grid is applied to the whole visual field. The binary union of features found at every possible location makes the extractor's features location-invariant. Each node represents the maximal value of its feature in the visual field. For example if feature 1 is present once or twenty times in the visual field X_1 will have the value *1*. The values of the *512* possible features form an input vector for the classifiers.

Noisy stimuli are generated using real-valued exponential noise with mean amplitude μ, which is added to the prototypical stimulus. The resulting noisy stimulus vector is given to the classifiers and their result is compared to its original label. All letters are tested and the results are combined to determine the percent of letters correctly identified for that noise level.

Multiple stimuli are generated by placing the prototypical stimuli in the visual field with a distance of greater than three pixels between the individual stimuli. This allows a simple union of prototypical input vectors. All possible N-letter combinations are tested. For example, if *N=2* there are *325* possible two letter combinations. The top *n* active nodes of each network are selected and compared to the original stimuli. The appropriate *n/n* histogram bin is incremented. For example, if *N=4* and the four most active letters are correctly matched, then the *4/4* histogram bin is incremented. If only *3* letters are matched, the *3/4* histogram bin is incremented.

The networks are setup or trained using the features extracted from the 26 prototypical letters. 26 output nodes are created: one for each letter.

A. Neural Network Setup

Both backpropagation and optimized learning methods are used to illustrate training issues.

'Naïve' NNs are trained via backpropagation with momentum. 100,000 training examples are randomly chosen with replacement from the prototypes. The number of hidden units used is increased until was sufficient to learn the input-output mapping to a component-wise tolerance of 0.05. The learning rate is .1, the momentum is 1 and there is a bias unit projecting to both hidden and output layers with value of 1. To improve 'Naïve' performance the NNs are retrained with noisy stimuli and multiple letter combinations. This is further described in the results section.

Optimized learning is performed using the most recent version of the *Waikato Environment for Knowledge Analysis* (WEKA) package currently available [28].

B. Regulatory Feedback Network Setup

To setup RFN, 26 output cells are defined. Each output cell is connected to input features that are active when its prototypical stimulus is displayed.

The RFN test phase involves an iterative algorithm. The more it is allowed to iterate, the more refined the outputs values can become. RFN is iterated until the maximum component-wise change between iterations is below a tolerance of 0.01. The starting activation value of each output is 0.01 for all trials.

RFN iterations can be optimized by eliminating the evaluation of 0-valued inputs and feedback units using sparse matrix techniques. Since the feature vector is often sparse, this results in speedups. Additional optimizations can be performed by removing output units which are zeroed (beyond the scope of this paper).

V. Results

The experiments test the networks on noisy and simultaneous presentations of the letters. This section evaluates the performance of the classifiers and methods to increase performance. Issues regarding setup of the networks and experiments are discussed in section *IV*.

In section *A*, the tolerance of the networks to random noise is tested. RFN has a good tolerance. However, with appropriate training and optimization NNs can outperform RFN. In sections *B, C* and *D* we test the networks for their ability to recognize simultaneous stimuli. With appropriate training and optimization NNs can perform almost as well as RFN on two simultaneous letters. However training and optimization is prohibitive in the four letter case. These limitations become even clearer when NNs must operate in multiple scenarios (as may be necessary in a brain). Thus, in section *E* we show that optimizing for noise affects multiple letter recognition and visa versa. This is significant because retraining a NN for every situation may not be as practical as an inherently more robust network (section *F*).

We begin by remarking that after setup and training described in section *IV*, both NNs and RFNs are able to accurately recognize all 26 letters individually. The resulting NNs had on average 12 hidden units.

A. Letters with Random noise

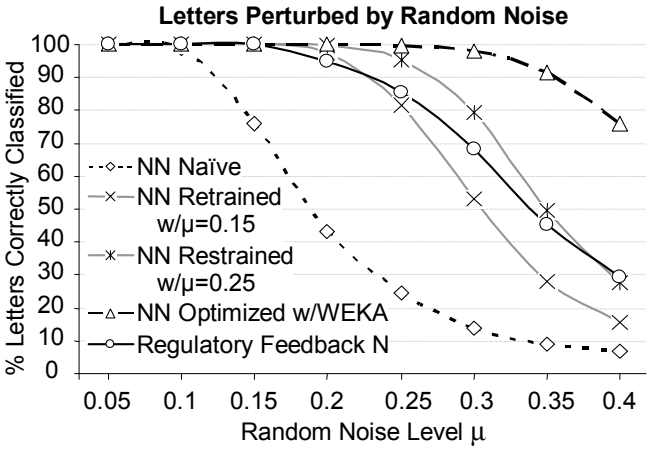

Figure 3: Tolerance of Random Noise. NN implemented w/WEKA is most tolerant. The Naïve NN is least tolerant. RFN is in-between. Tolerance of NN of highly depends on the choice of training set.

Given random noise, naive NNs (trained only on the 26 letters via backpropagation) perform sub-optimally compared to RFN. Yet, the RFNs set-up is even simpler.

Naive NNs achieve less than 70% correct responses when the mean exponential noise level μ is greater than *0.15*. RFNs achieve less than 70% correct responses only when μ is greater than *0.25*. See '*NN Naïve*' and '*Regulatory Feedback N*' in *figure 3*.

Retraining naïve NNs with noisy stimuli greatly improves their performance. Retraining requires 50,000 samples and the degree of improvement depends on the choice of training examples. For example retraining using stimuli with μ=0.15 does not increase performance as much as retraining with μ=0.25. Thus re-training stimuli must be carefully chosen. After such training NNs performance is similar to RFN.

NNs optimized using WEKA perform even better. This method is most optimal for random noise.

Since our goal is to evaluate RFNs in several scenarios as a general classifier, no attempt is made to optimize RFNs.

In summary, though RFN is not an ideal method for distinguishing stimuli in random noise, it performs this task well without optimization.

B. Two Letters Presented Simultaneously

RFN is better suited to disambiguate multiple stimuli. In this task naive NNs perform sub-optimally compared with RFN. Retraining and WEKA optimization improves NN performance almost to the level of RFN.

Naïve NNs correctly characterize 48 ± 3% of two letter combinations, and at least one letter in 96 ± 1% of the combinations. These statistics are determined with 9 NNs and calculated using *student's t* at the confidence level of *p=.05*. Though naïve NNs did not perform as well as RFNs they performed much better than chance: 0.4% two letters, and 17% for one letter out of two.

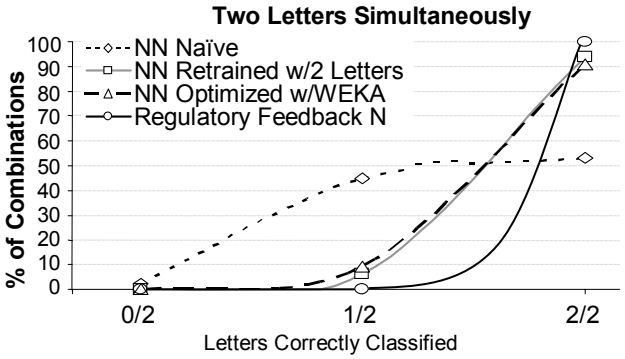

Figure 4: Two Letters Presented Simultaneously. RFN correctly determined all 2 letter combinations. NNs can be retrained and optimized achieve similar results.

NN retraining requires 100,000 samples randomly drawn from the set of all possible pairs of letters. Retraining significantly improves the NN's two-letter performance to 94% of 2/2 combinations. WEKA-optimized NNs also performed well, though slightly less at 91% of 2/2 combinations. See *figure 4*. The accuracy of NNs can possibly be increased further but this requires more training and resources.

In summary, RFN is the most ideal method tested here for distinguishing two simultaneous stimuli. This is bolstered by considering the simplicity involved in RFN setup.

C. Four Letters Presented Simultaneously

The networks are tested on all combinations of four letters simultaneously. The four most active nodes are evaluated for whether they match the four simultaneously presented stimuli. RFNs are able recognize correctly all 4 of the letters in 91% of the combinations. Naïve NNs perform poorly in this task and correctly identify only 3% of the combinations.

Retraining requires 100,000 samples drawn from the set of 14,950 possible four-letter combinations. Due to the number of combinations this is a significantly larger amount of training. The number training operations required increases in a combinatorial fashion with the number of letters presented simultaneously. Retrained NNs correctly identify 32% of 4 letter combinations. This is a marked improvement over naïve NNs and slightly better than the WEKA optimized NNs, which correctly identifies 28% of combinations. See *figure 5*.

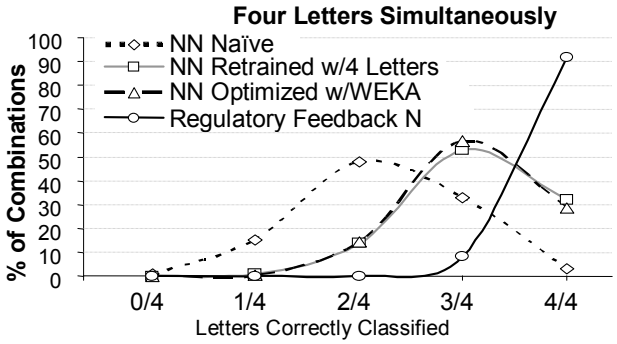

Figure 5: Four Letters Presented Simultaneously. RFN remains the most optimal method even when compared to retrained and optimized NNs. While retraining increased 4/4 performance, RFN achieved by far the most matches.

Thus, the RFN is the most ideal method tested here for distinguishing four simultaneous stimuli. Yet it is the simplest to set-up.

D. Five Letters

RFN are able to recognize correctly 79% of 5/5 letters from the 65780 possible combinations. This is astonishing given the amount of feature overlap in the inputs due to the multiple stimuli. Retraining and evaluation of NNs on five letters is not attempted because of the prohibitive number of examples required.

E. Retraining Can Reduce Robustness

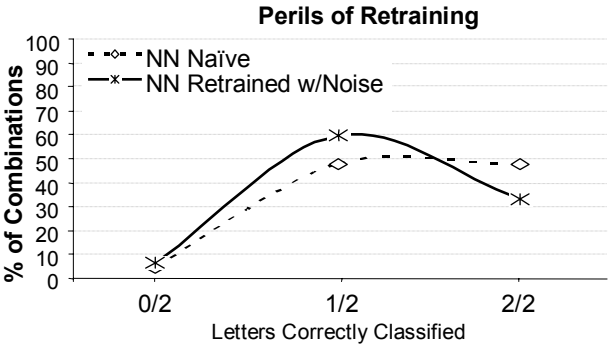

Figure 6: Retraining for Noise Can Decrease Multi-Letter Performance. After retraining naïve NNs for noise, the two letter performance is reduced. 2/2 matches decrease by 14% while 1/2 matches equivalently increase.

In many cases with retraining NN performance can be made comparable to RFN performance. However, when networks are trained for a particular task their performance in other tasks may decrease. Retraining a NN for the noise task can decrease its performance on the two-letter task. This is demonstrated in *figure 6*. After retraining naïve NNs on noisy stimuli, the composite of 9 re-trained networks show on average 47 less correct identifications of 2 letter combinations. This represents a 14% decrease in correct classifications ($p<<0.001$; $n=9$).

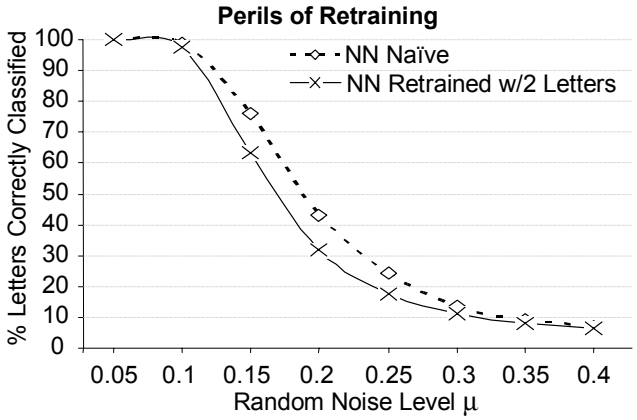

Figure 7: Retraining for Multiple Letters Can Decrease Tolerance to Noise. After retraining naïve NNs for two letters, noise performance is reduced. A smaller number of noisy letters are correctly classified at each noise level.

Similarly, retraining naïve NNs for the two letter task can decrease its performance in the noise task. *Figure 9* shows the decreased performance in a NN retrained for the two-letter task and tested for the noise task. To overcome this cross-interference, these scenarios would have to be carefully trained together which is not a simple problem (discussed further in section F). The innate robustness of RFN is a powerful way to avoid this issue.

F. Resource comparison: NN vs. RFN

In this section we compare the resources required for each network's set-up, connections, training phase, retraining phase and test phase.

RFN structure requires a sparse connectivity matrix for each output. The 'connection strengths' are determined by the number of feature nodes that are connected as inputs. Thus none of the 'connection strengths' are variable. The NN requires 12 hidden nodes and a bias node: subsequently 6494 variable weights.

The RFN was connected based on a single prototype for each letter. This requires only a single presentation. In contrast the naïve NN required up to 100,000 training cycles.

During the test phase, it takes NN's the equivalence of about 10,702 arithmetic operations to pass activation through its feed forward network regardless of stimulus difficulty. For single letters it takes RFN an average of 25,000 operations or 6 iterations to achieve less than a .01 change between iterations. For four simultaneous letters it takes an average of 90,000 operations or 23 iterations before reaching this criterion. RFNs may be interrupted at any point and in many cases the basic trends are observable within the first three iterations.

To achieve performance comparable to RFNs, NNs require various amounts of retraining based on the task. For two letters, 100,000 cycles were required for each of the 325 two letter combinations. For four letters, 100,000 cycles were required for each of the 14950 combinations. For noisy stimuli 50,000 cycles were required.

To be robust in all these tasks simultaneously, a naïve NN would have to be trained for all of them. The training sets would have to be carefully chosen and interwoven to avoid catastrophic interference [7,8]. Otherwise, NNs function can degrade as shown in figures 6 and 7.

Thus, inherent robustness may be more optimal than a training algorithm approach.

VI. Conclusion

In this section we summarize our results and discuss the significance of our findings in relation to natural scenarios.

Using a very simple set-up process, RFN is robust in multiple scenarios. RFNs are also computationally economical because they do not require many variables or a combinatorialy-implausible connectivity matrix [3].

In some scenarios such as stimulus perturbation by random noise, NNs can be trained to exceed RFN performance. However, random noise is not necessarily a very common classification scenario. For example, random noise may naturally occur during a fog or sandstorm.

On the other hand, scenes composed of multiple stimuli commonly occur in natural scenarios. Within these scenarios RFNs are most practical. Such scenarios can be described as occurring outside of the training distribution.

Organisms must maintain flexibility and cope with these scenarios. For example, an animal which has only seen single predators may encounter two predators at the same time. That animal will not have time for retraining.

Not only can NN training requirements limit the network's ability to generalize, the NN training strategy requires carefully planned training protocols. Since retraining on one scenario can degrade previous training on another, NN retraining for multiple simultaneous scenarios are even more likely to become complex and combinatorially intractable. As the number of classes and features or the complexity of the environment grows, the number of training operations grows nonlinearly. Thus plausible NN training requires a combinatorially intractable amount of retraining.

Furthermore, backprop-like learning signals are yet to be found within biological networks. Thus, from a biological and computational perspective RFN may be a more plausible model of the brain [2].

NN training requirements may vary with optimized training algorithms such as: quickprop, resilient backprop and conjugate gradient (i.e. [29]). However, such algorithms require other constraints and do not address the exploding complexity required by the training set.

Yet NNs are important. RFN are not suited for all applications. For example, since RFNs function outside of the training distribution, RFNs are not universal approximators. Thus they may not be able to imitate any arbitrary function like NNs can.

In conclusion, regulatory feedback networks represent an important approach to classification. RFNs require fewer resources and are more robust. Even a structure that is only slightly better able to generalize without retraining can conserve many computational resources. Thus, studies to gain insight about inherent robustness are very important.

References

[1] Achler T, (2002) Input Shunt Networks, Neurocomputing, 44-46c: 249-255.

[2] Achler T, (2007) Object classification with recurrent feedback neural networks, Proc. SPIE, Evolutionary and Bio-inspired Computation: Theory and Applications, Vol. 6563.

[3] Achler, T., Amir, E. (2008) Input Feedback Networks: Classification and Inference Based on Network Structure, Artificial General Intelligence Proceedings V1: 15-26

[4] Hinton G. E, Sejnowski T. J. (1986) Learning and Relearning in Boltzmann Machines. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vl 1 MIT

[5] Rosenblatt, F. (1958) The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Psychological Review, V65.

[6] Rumelhart, D. E., & McClelland, J. L. (1986). On learning the past tenses of English verbs. In J. L. McClelland & D. E. Rumelhart (Eds.), Parallel distributed processing (V2, p216-271). MIT Press.

[7] McCloskey, M. & Cohen, N.J. (1989) Catastrophic interference in connectionist networks: The sequential learning problem. In The Psychology of Learning and Motivation, V24. NY Acad, p109–165.

[8] Sharkey, N.E. & Sharkey, A.J.C. (1995) An analysis of catastrophic interference. Connection Science, 7, 301–329.

[9] Turrigiano G G, Nelson S B. (2004) Homeostatic Plasticity In The Developing Nervous System, Nature Reviews Neuroscience 5, 97-107

[10] Marder E, Goaillard JM. (2006) Variability, compensation and homeostasis in neuron and network function. Nat Rev Neurosci, 7(7):563-74.

[11] M. Sugiyama, Active (2006) Learning in Approximately Linear Regression Based on Conditional Expectation of Generalization Error, Journal of Machine Learning Research, 141-166:7.

[12] Marcus, G.F. (1998) Rethinking eliminative connectionism Cognit. Psychol. 37, 243—282.

[13] Elman, J.L. (1998). Generalization, simple recurrent networks, and the emergence of structure. In M.A. Gernsbacher and S.J. Derry (Eds.) Proceedings of the Twentieth Annual Conference of the Cognitive Science Society.

[14] Charles, J. J., L. I. Kuncheva, B. Wells, and I. S. Lim. 2006. An evaluation measure of image segmentation based on object centres. Image Analysis and Recognition, Pt 1 4141:283-294.

[15] van der Velde, F., de Kamps M., (2001) From Knowing What to Knowing Where: Modeling Object-Based Attention with Feedback Disinhibition of Activation, Journal of Cognitive Neuroscience, 13(4), 479-491

[16] Herd, S.A. & O'Reilly, R.C. (2005). Serial visual search from a parallel model. Vision Research, 45, 2987-2992.

[17] Duncan, J. and G. W. Humphreys (1989). "Visual-Search and Stimulus Similarity." Psychological Review 96(3): 433-458.

[18] Reggia, J. A., Dautrechy C. L., (1992). "A Competitive Distribution-Theory of Neocortical Dynamics." Neural Computation 4(3): 287-317.

[19] LaBerge, D. (1997) "Attention, Awareness, and the Triangular Circuit". Consciousness and Cognition, 6, 149-181

[20] Atema J, (1998) Distribution of Chemical Stimuli, in Sensory Biology Of Aquatic Animals, Atema J, Fay RR, Proper AN & Tavolga eds, p 29-56. Springer-Verlag, NY

[21] Chen WR, Xiong W, & Shepherd GM, (2000) Analysis of relations between NMDA receptors and GABA release at olfactory bulb reciprocal synapses, Neuron, 25:625–633.

[22] Aroniadou-Anderjaska V, Zhou F-M, Priest CA, Ennis M, & Shipley MT, (2000) Tonic and synaptically evoked presynaptic inhibition of sensory input to the rat olfactory bulb via GABAB heteroreceptors. J Neurophysiol, 84: 1194–1203.

[23] Isaacson J S. and Strowbridge B W (1998) Olfactory Reciprocal Synapses: Dendritic Signaling in the CNS, Neuron, Vol. 20, 749–761.
[24] Lowe, G. (2002) Inhibition of backpropagating action potentials in mitral cell secondary dendrites. J Neurophysiol 88: 64–85.
[25] McFadden F E. (1995) Convergence of Competitive Activation Models Based on Virtual Lateral Inhibition, Nrl Ntwrks V8#6
[26] Nagle, Saff, Snider (2000) Fundamentals of Differential Equations and Boundary Value Problems, 3rd ed, Ch13.
[27] J. S. Sanchez, L.I. Kuncheva (2007) Data reduction using classifier ensembles, Proc. 11th European Symposium on Artificial Neural Networks
[28] Ian H. Witten and Eibe Frank (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005. Software available at http://www.cs.waikato.ac.nz/ml/weka/ (version 3.5.6)
[29] Fahlman, S.E. (1989), "Faster-Learning Variations on Back-Propagation: An Empirical Study", in Touretzky, D., Hinton, G, and Sejnowski, T., eds., Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann, 38-51.