Symbolic neural networks for cognitive capacities

Tsvi Achler

ITOP Corporation, 3168 South Court, Palo Alto, CA 94306, United States

Received 19 April 2014; received in revised form 1 July 2014; accepted 3 July 2014

Abstract
Pattern recognition (recognizing a pattern from inputs) and recall (describing or predicting the inputs associated with a recognizable pattern) are essential for neural-symbolic processing and cognitive capacities. Without them the brain cannot interact with the world e.g.: form internal representations and recall memory upon which it can perform logic and reason. Neural networks are efficient, biologically plausible algorithms that can perform large scale recognition. However, most neural network models of recognition perform recognition but not recall: they are sub-symbolic. It remains difficult to connect models of recognition with models of logic and emulate fundamental brain functions, because of the symbolic recall limitation.

I introduce a completely symbolic neural network that is similar in function to standard feedforward neural networks but uses feedforward-feedback connections similar to auto-associative networks. However, unlike auto-associative networks, the symmetrical feedback connections are inhibitory not excitatory. Initially it may seem counterintuitive that recognition can even occur because the top-down connections are self-inhibitory. The self-inhibitory configuration is used to implement a gradient descent mechanism that functions during recognition not learning. The purpose of the gradient-descent is not to learn weights (weights are still learned during learning) but to find neuron activation. The advantage of this approach is the weights can now be symbolic (representing prototypes of expected patterns) allowing recall within neural networks. Moreover, considering the costs of both learning and recognition, this approach may be more efficient than feedforward recognition. I show that this model mathematically emulates standard feedforward model equations in single layer networks without hidden units. Comparisons that include more layers are planned in the future.

© 2014 Elsevier B.V. All rights reserved.
Introduction

The neural networks responsible for pattern recognition determine the form of information and connection weights required for recognition, memory, and further processing. However, certain forms may be more optimal for certain tasks than others. Popular feedforward neural networks are efficient recognition algorithms however are not optimal for recall: describing or predicting the inputs associated with a recognizable pattern. Thus feedforward networks are considered sub-symbolic (e.g. Fodor & Pylyshyn, 1988; Sun, 2002). In this work we will discuss and show how it possible to change the form of information and achieve both recognition and recall with symbolic neural networks. Moreover, our current simulations, limited single layer, show symbolic networks may be even more efficient for recognition than feedforward networks.

First let us define the meaning of symbolic. A symbolic connection can be thought of as the relationship between an input and the output node that does not depend on any other inputs and outputs. For example, suppose there is an input node that represents legs and an output node that represents zebra. The symbolic relationship for zebra and legs is that it has 4 legs. In a symbolic network, the value 4 can be the weight that represents the connection strength between legs and zebra. This represents a prototype description of a zebra (node) where it does not matter whether there exist other animals (other nodes) that also use the same input (and may also have 4 legs such as dogs, cats, rats, and giraffes, or may have none, 2, 6, 8 or any other number legs). The symbolic weight between zebra and legs remain the same regardless of other nodes.

However, it is important for recognition that certain information will be unique. For example, if the only information available at the input is 4 legs, it would not be possible to recognize a zebra from any other animal with 4 legs, regardless of the type of network. Unique information (such as distinctive stripes) is needed to properly perform recognition regardless whether the network is feedforward or symbolic.

A major difference between feedforward networks and the proposed symbolic network is how uniqueness is evaluated and encoded. Feedforward networks solve recognition with one multiplication per layer. In order to recognize correctly within one multiplication, they encode uniqueness information in the weights (more than just symbolic information). To incorporate the uniqueness information into weights, feedforward methods use a gradient descent error-driven mechanism during learning, which rehearses the patterns of the training set in a uniform identical error-driven mechanism during learning, which rehearses information). Thus feedforward networks are considered sub-symbolic (e.g. Fodor & Pylyshyn, 1988; Sun, 2002). In this work we will discuss and show how it possible to change the form of information and achieve both recognition and recall with symbolic neural networks. Moreover, our current simulations, limited single layer, show symbolic networks may be even more efficient for recognition than feedforward networks.

By definition, symbolic weights cannot incorporate whether information is unique, because uniqueness depends on whether other nodes use that information (and by definition symbolic information must be independent of other outputs).

However, it is necessary to determine uniqueness for recognition, thus the proposed model uses a gradient descent mechanism during recognition to determine how a unique piece of information is based on the other nodes that are currently active (e.g. the other animals that are also being considered) and modulates the relevance of the input (e.g. stripes) accordingly. In effect the proposed model is doing what feedforward learning algorithms do during learning (modulate weights based on uniqueness) but during recognition (modulating inputs based on uniqueness). The gradient descent of the proposed model does not learn weights (weights do not change) but determines uniqueness.

The symbolic model does not require a gradient descent mechanism during learning and subsequently its learning is much easier. Moreover, during recognition the current test pattern is available (while not available during learning). This translates into better efficiency. Feedforward learning algorithms have to perform a gradient descent using the all the patterns in the training set (in iid form) and determine overall how relevant each input is. In contrast, the proposed model only performs a gradient descent using the current input pattern. This requires much less gradient descent iterations, iteration times, and allows a simpler, quicker and recallable form of learning.

Clearly our zebra example is a gross oversimplification; there are many types of legs and features that go into legs and so on. However, this generalizes to more complex networks. Networks that can be described in a feedforward manner (including multilayer networks) can also be described in a symbolic manner. Even if nodes are hidden, they can still have symbolic weights. The difference is in our symbolic network uniqueness information is not incorporated into weights. Our analysis in this paper is currently limited to single layer networks, but the symbolic properties are generalizable to hidden nodes as well. The differences between the networks will hopefully become clearer in the examples.

Let us more formally establish the standard notation for neural networks and pattern recognition on which we shall build, and then review and compare neural network models. Let vector \( Y \) represent the activity of a set of labeled nodes that may be called output neurons, or classes and individually written as \( Y = (y_1, y_2, y_3, \ldots, y_h) \). Supervised neurons identify patterns using labels or guide behavior. “Supervised” tasks are explicitly naming patterns or behaviorally responding to the environment (without explicit naming). The definition of supervised includes any representation that is behaviorally relevant or action-able. This definition may be a broader than other authors’ definitions. For example labels can include: y’s representing “dog, cat”, predator, prey, or food, mate, danger, and so on. Without labeled associations the brain cannot interact with the world, for example find: food, mates, and hazards. All of these must be correctly labeled by the organism (whether they are explicitly named or not) and responded with the appropriate behavior. Thus supervised recognition is a broad and essential foundation
Symbolic neural networks for cognitive capacities

upon which survival, cognition, and intelligence are based. If the meaning of the Y’s are not explicitly defined or behaviorally linked (for example, the search for efficient representations determines the Y vectors without behaviorally-linked labels) then the nodes are considered unsupervised. However, once a node becomes behaviorally relevant even if it was determined using a mechanism optimizing efficiency, it is defined as supervised. Note that the definition of supervised/labeled is independent of the definition of symbolic.

Finally, vector X represents sensory nodes that sample the environment, or input space to be recognized, and are composed of individual features \( X = (x_1, x_2, x_3, \ldots, x_n) \) which form the basis of recognition. The input features can be sensors that detect edges, lines, frequencies, kernel features, and so on, or may be determined by supervised methods.

Now let us establish a notation for recall. Assume neuron \( Y_1 \) is most-optimally tuned to an input pattern "A". Let us label this supervised neuron using its label and optimal pattern and write it as neuron \( y_A \). We describe its optimal input pattern as \( X_a \) where pattern "A" which is represented by the feature sensory nodes with values \( X_a = (x_{a1}, x_{a2}, x_{a3}, \ldots, x_{an}) \). This definition is important to distinguish the proposed network from other networks and establish the underlying expectations of neurons. Neural networks are not commonly defined by a recall pattern, because most models are sub-symbolic and such information is not readily available or recallable (Fodor & Pylyshyn, 1988; Sun, 2002).

We will demonstrate this further, but first let us review and distinguish between neural network models for example: Feedforward, Lateral Inhibition, Auto-Associative, etc.

Based on feedforward weights \( W \), feedforward neural network models solve the recognition relationship:

\[
\hat{Y} = W \hat{X} \quad \text{or} \quad \hat{Y} = f(W, \hat{X}) \quad (1)
\]

These neural networks represent a feedforward connectionist architecture because the direction of information flow during recognition is feedforward: one-way from inputs to the outputs. \( W \) represents a feedforward matrix of weights or parameters that associates inputs and outputs. \( W \hat{X} \) calculates the output using the feedforward weights and inputs. Feedforward networks can be found in the literature with different algorithm optimizations, for example: single-layer Perceptrons (Rosenblatt, 1958), multilayer Neural Networks with nonlinearities introduced into calculation of \( \hat{Y} \) (Rumelhart & McClelland, 1986), and machine learning methods such as 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 that in recognition involves a feedforward transformation using \( W \) during recognition.

Some feedforward algorithms include lateral connections for competition between output nodes \( Y \), during recognition. Lateral connections can be designated as one-vs.-all, winner-take-all, all-vs.-all e.g. (Rifkin & Klautau, 2004). However, these competition methods still rely on initially calculating \( Y \) node activities based on feedforward-trained weights \( W \).

Recurrent networks are feedforward networks where a limited number of outputs are also used as inputs. However, the outputs do not modify their own inputs, thus can be unfolded into a finite feedforward network. This allows the processing of time e.g. (Schmidhuber, 1992; Williams & Zipser, 1994). But from a computational perspective they still remain feedforward networks. Similarly, other models also learn signals delayed over time and result in feedforward networks (e.g. Rao & Huang, 2011; van Hateren & Ruderman, 1998).

Unsupervised models are important because they can determine hidden (unsupervised) units important for efficient recognition. Development of efficient hidden nodes within the model is planned in the future in conjunction with the introduction of hierarchy. Unsupervised feedforward models include deep learning (Hinton, Osindero, & Teh, 2006) and predictive coding networks (Olshausen & Field, 1996; Rao & Ballard, 1999; Rao & Huang, 2011). It is important to note that even though Hinton’s Deep Learning with the Restricted Boltzmann Machines (RBM) and contrastive divergence (CD) learning algorithm (Hinton, 2002; Hinton et al., 2006) and others (such as backprop) are often considered feedforward-feedback because they rely on feedforward-feedback connections, they do so during learning. However, after they learn feedforward weights, they are subsequently feedforward models during recognition since they use Eq. (1) for recognition (also see Table 1). This paper focuses on supervised models without hidden nodes on feedforward-feedback models.

Several supervised models use feedforward-feedback connections during recognition where the outputs feed back to the same inputs that activated them. One is Adaptive Resonance Theory ART (Carpenter & Grossberg, 1987) which implements a mechanism to measure a goodness-of-fit using top-down connections after a winner-take-all (WTA) competition. WTA requires an expansive network-wide inhibition of all of the outputs to allow only a single output to be active. Because of this inhibition, a drawback to WTA and subsequently ART is that it requires the brain to stop and evaluate possible patterns individually, store a goodness of fit measure for each pattern, cycle through the rest of the patterns, and decide which pattern received the best fit. There is little evidence suggesting the brain or brain tissues have only one output active a time (winner-take-all) throughout every cycle of the evaluation process.

Auto-associative models have symmetrical (same weight) feedforward and feedback connections (from input to output and output to input) and perform recognition. Hopfield networks (e.g. Hopfield, 1982) perform unsupervised recognition and Bidirectional Associative Memory (BAM) networks (Kosko, 1988) perform supervised recognition. Auto-associative networks have binary connections and binary or binary-like (sigmoid function) neuron activation. This is because auto-associative feedback is positive (excitatory) and a property of this is that iterative activation of even small values will lead to the maximal values regardless whether non-binary values are used.

In the model proposed, the “auto-associative” symmetrical feedback is negative (inhibitory). The connections still have symmetrical weights but the feedback weights are inhibitory. Thus it can be called auto-dissociative (although this name sounds more like a personality disorder). I have been calling it regulatory feedback or iterative neural networks to emphasize the feedforward-feedback and inhibitory properties during recognition. Neither the activation
nor the weights are limited to binary values. Self-inhibitory connections for recognition may initially seem counterintuitive, however these connections implement a gradient descent mechanism that converges to and will be shown mathematically to perform equivalent recognition using symbolic weights as feedforward networks using feedforward weights. Moreover, the overall network can be shown to work with either subtractive inhibition or shunting (dividing) inhibition.

The most-similar networks in the literature to the proposed method (with shunting inhibition) are virtual lateral inhibition networks (Reggia, D’Autrechy, Sutton, & Weinrich, 1992). In their pioneering work, it is observed that feedforward-feedback connections of this form can cause inhibition networks (Reggia, D’Autrechy, Sutton, & Weinrich, 1992). In their pioneering work, it is observed that feedforward-feedback connections of this form can cause competition that may not be easily distinguished from lateral inhibition. However, the roles of symbolic weights, recall and recognition, and mathematical relationship between feedforward and feedforward-feedback networks have not been described.

The model presented is a supervised recognition algorithm that uses feedforward-feedback configuration to implement a gradient descent mechanism during recognition. This allows learning to be much simpler using symbolic-like expectations where the new weights (M) represent prototypes, see Table 1.

### Mathematical formulation

The recognition method proposed here can be considered interchangeable with supervised feedforward neural networks because they ultimately solve the same equation: Eq. (1). However, it does not require explicitly finding W and instead uses symbolic weights. In fact, it can be shown that W is the pseudoinverse of symbolic weights. Eq. (2) can be rewritten from Eq. (1):

\[ W^{-1} \mathbf{y} = \mathbf{x} \]  

Let us define matrix M as the inverse or pseudoinverse of matrix W. The relation becomes:

\[ M \mathbf{y} - \mathbf{x} = 0 \]  

To demonstrate that matrix M represents the expectation, let us write Eq. (3) using matrix elements.

\[
\begin{bmatrix}
m_{11} & \cdots & m_{1k} \\
\vdots & \ddots & \vdots \\
m_{j1} & \cdots & m_{jk}
\end{bmatrix}
\begin{bmatrix}
y_1 \\
\vdots \\
y_s
\end{bmatrix} -
\begin{bmatrix}
x_1 \\
\vdots \\
x_j
\end{bmatrix} =
\begin{bmatrix}
0_1 \\
\vdots \\
0_s
\end{bmatrix}
\]

Let us suppose \( \mathbf{x} = [x_1, x_2 \ldots x_j] \) is an expected pattern when a single node is active. Let us activate one node (node 1 in this case \( \mathbf{y} = [1 \ 0 \ldots 0]^T \)) and present the expectation pattern \( \mathbf{x} = [x_1, x_2 \ldots x_j]^T \).

Table 1 Comparisons between feedforward and feedforward-feedback supervised classifiers.

<table>
<thead>
<tr>
<th>Structure during recognition</th>
<th>Dynamics during</th>
<th>Weights</th>
<th>Symbolic relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feedforward</td>
<td>Learning</td>
<td>W</td>
<td>Sub-symbolic</td>
</tr>
<tr>
<td>Feedforward-Feedback</td>
<td>Recognition</td>
<td>M</td>
<td>Symbolic</td>
</tr>
</tbody>
</table>

We get \( [m_{11}, \ldots, m_{1j}]^T = [x_1 \ldots x_j]^T \). The weights corresponding to node 1 \( [m_{11}, \ldots, m_{1j}]^T \) are the expected symbolic pattern in X. This is true for any other vectors of M as well. Thus the vectors of M are the expected symbolic values of corresponding patterns in X and M is a matrix of symbolic weights.

We showed that the weights are symbolic now let us outline how this connectionist method works during recognition. When output nodes in Y are active, a pattern is generated from stored weights in M. The generated pattern is MY and this pattern is subtracted from the incoming input pattern X (through inhibition). When a correct recognition is made, the stored pattern that is activated, MY, will account for the input pattern, X, and they will annihilate each other making the sum 0, see Eq. (3). This shares some similarity to other generative models however the difference is this process occurs during recognition NOT during learning: to find activation not to learn weights.

Returning to our derivation, this account is not complete. Although Eq. (3) describes the basic relationship between inputs X, output activation Y, and feedback weights M, it does not provide a way to project input information to the outputs and determine activation. In other words, in this form it is not possible to calculate Y given X and M without calculating the pseudo-inverse of M.

To avoid calculating the pseudoinverse and W, we use a gradient descent mechanism using the auto-inhibitive connections to converge to Eq. (3). We define the final solution as an energy function to be minimized, and then use iterations to get to that point.

To do that let us rewrite Eq. (3) in terms of error E.

\[ E_{\text{method}} = M \mathbf{y} - \mathbf{x} \]  

When error goes to zero \( E = 0 \), Eq. (8) becomes Eq. (3), where \( \mathbf{x} = M \mathbf{y} \) and also solves Eq. (1). Taking the derivative of E relative to Y the equation becomes:

\[ dE/dY = -M^T(E) = -M^T(\mathbf{x} - M \mathbf{y}) \]
Elaborating further on Eq. (9), if \( \dot{X} \) is held constant during recognition then \( dE \) is directly proportional to \( dY \). Since the derivative of energy is linearly proportional with respect to the derivative of \( Y \) we can write the whole equation in terms of \( dY \). \( dE \) is substituted with \( dY / dt \) as follows:

\[
\frac{dY}{dt} = -M^T (MY - X)
\]  

(10)

This now describes self-contained behavior of neurons in a network which minimizes energy to find activation. Looking at this equation it is clear that values of \( Y \) that satisfy \( MY - X \) (solving the same problem as \( Y = WX \)) will be stable \( (dY = 0) \). Values of \( Y \) that will cause \( MY \) to be greater than \( X \) will be decreased \( (dY < 0) \) and values of \( Y \) that generates a pattern \( (MY) \) that is less than \( X \) will be increased \( (dY > 0) \). In other words, solutions that satisfy \( MY - X \) are fixed points. Neurons in a network behaving according to this equation will minimize error and in the process determine neuronal activity. Iterating this equation until steady state \( (dY/dt = 0) \) during recognition (with no weights adjusted) results in a solution that is equivalent to \( Y = WX \). Both symmetrical feedforward and feedback connections are designated by \( M \); \( MY \) transforms \( Y \) information into the \( X \) domain, thus a feedback process; \( M^T \) transforms \( X \) information into the \( Y \) domain, thus a feedforward process.

This method is based on least squares to minimize the error, looks similar to a predictive coding algorithm which performs a gradient descent to minimize energy, has bidirectional connections, iterative function, and has generative properties (e.g. Rao & Huang, 2011; Olshausen & Field, 1996; Rao & Ballard, 1999). Unlike a learning algorithm, the gradient descent here occurs during recognition and weights here are not learned via this gradient descent. Instead activation \( Y \) is determined by gradient descent and there is no sparseness term. To repeat, learning is defined as the phase when weights change. Minimization of error is not synonymous with learning. This model shows it is possible to do minimization of error by changing activation without changing weights. Minimizing energy during recognition finds activation \( Y \), the same \( Y \) that is found using feedforward weights \( W \) in Eq. (1), but using \( M \). This allows simple symbolic weights to be used.

In this model only the currently presented pattern needs to be iterated. In feedforward learning, all training patterns need to be repeatedly rehearsed in independent and identically distributed form (iid) for learning. This is why the overall computational costs in this model appear less compared to feedforward learning. See algorithm comparison demonstration (Section ‘Learning Example’).

This model can take other forms as well. Convergence to the same characteristic vectors can be achieved using divisive inhibition. For example, the inhibition described by Eqs. (8)–(10) is subtractive but it is possible that inhibition is divisive or shunting. There is evidence that biological neural inhibitors such as GABA chlorine channels behave more like a divisive inhibitor than a subtractive inhibitor.

Using shunting inhibition, let us redefine the error as a ratio:

\[
\tilde{E}_{model2} = \frac{X}{MY}
\]

(11)

Using Eq. (11) as the basis of evaluating error, \( \tilde{E} = 1 \) represents the outputs correctly matching the inputs with no error and the values of all parameters \( (M, Y, X) \) are positive real values. This alternative is important because the subtractive dynamics are more sensitive to step sizes \( dt \) than the divisive method. It also shows that a variety of neural types can achieve a similar gradient descent mechanism (Achler & Bettencourt, 2011).

The shunting equation can be written as:

\[
\frac{d\tilde{Y}}{dt} = \tilde{Y} \left( \frac{1}{V} M^T \tilde{E}_{model2} - 1 \right)
\]

\[
= \tilde{Y} \left( \frac{1}{V} M^T \left( \frac{X}{MY} \right) - 1 \right) \quad \text{where} \quad V = \sum_{j=1}^{N} M_{ji}
\]

(12)

Using alternative notation, this can be written as:

\[
\frac{dy_i}{dt} = \frac{-y_i}{\sum_{j=1}^{N} m_{ji}} \sum_{k=1}^{N} m_{ki} \left( \frac{x_k}{\sum_{b=1}^{N} m_{ib}y_b} \right) - y_i
\]

(13)

where \( M_{naff} \) are the dimensions of \( M \). Both models have identical fixed point solutions when \( dY/dt \) converges to zero (Achler & Bettencourt, 2011). Previous work has also shown that a Lyapunov function can be written for an inclusive class of equations (to which Eq. (13) belongs) and is guaranteed to converge to \( dY/dt \rightarrow 0 \) (Achler, 2011, chap. 4; Mcfadden, Peng, & Reggia, 1993). Other variants of gradient descent (beyond using subtractive and shunting inhibition) are possible as well e.g. (Wilkinson, 1999, chap. 9).

Demonstrations

The goal of the examples given here is to show in detail why this model works and how symbolic information can be used. Thus examples are quite simple and based on a single layer without hidden nodes. Most of the examples are based on two nodes. A discussion of multiple nodes can be found in Section ‘Multiclass classification’ including references to results.

The first example shows the simplest network, the differences between \( M \) and \( W \) and the steps of gradient descent. The second example reinforces the claim that feedback weights \( M \) are symbolic while feedforward weights \( W \) are sub-symbolic. The third example shows that generalized learning can be achieved intuitively using \( M \) and compares computational resources. The fourth describes multiclass classification findings.

Gradient descent during recognition

A simple two node configuration (Fig. 1, example 1) is presented to illustrate the basic workings and dynamics of the model. Let us say there are two inputs \( x_1 \) and \( x_2 \) and we
Fig. 1 Feedforward-feedback neural configuration determined by $M$ of Eq. (16). The feedforward-feedback connections are symmetric (equal) but the feedback weight inhibits. Connections with weights of 0 are not drawn.

want a neural network where one neuron becomes active when $x_1 = 1$ and $x_2 = 0$. We want the other neuron to be active when $x_1 = 1$ and $x_2 = 1$. These expectations can be summarized in a matrix form shown below.

$$ \text{Expectation} = \begin{bmatrix} x_1 & x_2 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad (14) $$

If we were to determine a conventional feedforward network, we would run a supervised training algorithm on these patterns to learn $W$ through a gradient descent learning algorithm. Training patterns will be presented patterns in iid form with fixed frequency and random order until the network converges to the correct $W$ which is:

$$ W = \begin{bmatrix} x_1 & x_2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad (15) $$

Given these feedforward weights $W$ we can now calculate $\hat{Y} = WX$ and get the correct solution. However, given only $W$ without knowing the original criteria we cannot tell what was the original expectation and symbolic information. For example is neuron $y_2$ optimized for $x_1 = 0$ and $x_2 = 1$?

Let us compare with the symbolic method. Since we described the network through its expectations, and $M$ represents expectations we can simply write $M$ as follows:

$$ M = \begin{bmatrix} x_1 & x_2 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad (16) $$

$M$ determines the feedforward-feedback network connectivity (see Fig. 1). In the configuration above, 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 ($x_1$ and $x_2$).

Using Eq. (10) or (13) and $M$ designated in (15), the algorithm can be numerically iterated (see Table 2). However, for this specific simple case we can mathematically solve the equations. Thus we solve Eq. (13) with the given $M$ to demonstrate the solution. Let us start with writing E:

$$ e_1 = x_1 - \frac{y_1}{C_1} = \frac{x_1}{y_1 + y_2} \quad (17) $$

$$ e_2 = x_2 - \frac{y_2}{C_1} = \frac{x_2}{y_1 + y_2} \quad (18) $$

Now let us place this into Eq. (13):

$$ \frac{dy_1}{dt} = \frac{y_1}{1} (1 \cdot e_1 + 0 \cdot e_2) - y_1 = \frac{y_1 \cdot x_1}{y_1 + y_2} - y_1 \quad (19) $$

$$ \frac{dy_2}{dt} = \frac{y_2}{2} (1 \cdot e_1 + 1 \cdot e_2) - y_2 = \frac{y_2}{2} \left( \frac{x_1 + x_2}{y_1 + y_2} \right) - y_2 \quad (20) $$

Eqs. (19) and (20) will be used to derive the steady state solutions. Let us rewrite these equations in terms of $Y_{\text{new}} = Y_{\text{old}} + dY = f(Y_{\text{old}})$:

$$ Y_{1,\text{new}} = Y_1 + \frac{dy_1}{dt} = \frac{y_1 + x_1}{y_1 + y_2} \quad (21) $$

$$ Y_{2,\text{new}} = Y_2 + \frac{dy_2}{dt} = \frac{y_2 + x_2}{y_1 + y_2} \quad (22) $$

---

**Table 2** Comparison of the difference in pseudo-code between single layer feedforward and symbolic networks.

<table>
<thead>
<tr>
<th>Neural network</th>
<th>During learning</th>
<th>During recognition (given $X_{\text{test}}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-symbolic</td>
<td>1. Organize supervised training set in iid order</td>
<td>1. Calculate: $Y = WX_{\text{test}}$</td>
</tr>
<tr>
<td>Feedforward (during recognition)</td>
<td>2. Initialize network with number of nodes and inputs</td>
<td>2. Iterate until $dY &lt; \text{threshold}$</td>
</tr>
<tr>
<td>Feedforward-feedback (during learning)</td>
<td>3. Iterate until learning error $&lt; \text{threshold}$</td>
<td>i. Present an iid pattern from training set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ii. Calculate delta and error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>iii. Update weights $W$</td>
</tr>
<tr>
<td>Symbolic</td>
<td>1. Average the training set using the supervised labels: $M = \mu(X</td>
<td>Y)$</td>
</tr>
<tr>
<td>Feedforward-feedback (during recognition)</td>
<td>2. Iterate until $dY &lt; \text{threshold}$</td>
<td>2. Iterate until $dY &lt; \text{threshold}$</td>
</tr>
<tr>
<td>Feedforward (during learning)</td>
<td></td>
<td>i. Calculate: $dY = -M^T(M\hat{Y} - X_{\text{test}})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ii. $Y_{\text{new}} = \hat{Y} + dY$ or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>iii. $Y_{\text{new}} = \hat{Y} + dY = \left( \sum_{j = 1}^{M} M_j^T \frac{X_{\text{test}}}{M \cdot Y} \right)$</td>
</tr>
</tbody>
</table>

---

76 T. Achler
Let us show the gradient descent dynamics during testing given \( x_1 = 1, x_2 = 1 \), and initial conditions \( y_1 = 0.1, y_2 = 0.1 \) and \( y_2 = 0.1 \) (see Fig. 2). A simulation showing iterations toward steady-state yields an intuition about the algorithm. The example is shown for \( x_1 = x_2 = 1, \) with the initial activity of \( y_1 \) and \( y_2 \) assumed to be 0.1 (\( y_1 = 0.1 \) and \( y_2 = 0.1 \) at \( t < 1 \)).

\( t = 0 \) represents the initial condition. At \( t = 1b \) the activity of \( y_1 \) and \( y_2 \) are projected back to the inputs. Both \( e_1 \) and \( e_2 \) are large because representations \( y_1 \) and \( y_2 \) that use \( x_1 \) and \( x_2 \) are not very active (initial values are 0.1). Thus the feedforward signal \(( e_1 \) and \( e_2 \)) is boosted. Note that \( e_2 \) is larger than \( e_1 \), because two nodes use the input associated with \( e_2 \). At \( t = 1f \) the error is 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 normalized by the number of node processes and the error. At \( t = 2b \) the activity of \( y_1 \) and \( y_2 \) are projected back to the inputs again. This time the output nodes are more active and produce a more accurate reconstruction so error values \( x_1 \) and \( x_2 \) are smaller. From \( t = 2f \) to \( t \rightarrow \infty \) 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 steady state results when \( x_1 = x_2 = 1 \) is \( y_1 = 0 \) and \( y_2 = 1 \).

The solutions at steady state can be derived for this simple example by setting \( dy_1/dt = 0 \) and \( dy_2/dt = 0 \) and solving the equations. The solutions are:

\[
y_1 = x_1 - x_2 \quad \text{and} \quad y_2 = x_2 \quad \text{if} \quad x_1 > x_2
\]

Eq. (7) or (9) cannot have negative values thus if \( x_1 \leq x_2 \) then \( y_1 = 0 \) and the equation for \( y_2 \) is determined by:

\[
y_2 = \frac{x_1 + x_2}{2}
\]

Making \( x_1 = x_2 = 1 \) the solution is \( y_1 = 0 \) and \( y_2 = 1 \) as in the simulation. If \( x_1 = 1 \) and \( x_2 = 0 \) then \( y_1 = 1 \) and \( y_2 = 0 \). Note this solution reflects the entries in \( M \), since \( M \) represents the solutions. This greatly simplifies recognition and makes the network symbolic. In the next few examples we show the symbolic properties.

**Example of a more complex symbolic recognition and recall**

Here we compare feedforward supervised recognition with our generative method using a simple symbolic recognition problem. We show the weights in \( M \) are more intuitive, easier to learn, than \( W \) and are symbolic.

Suppose we want to discriminate between idealized drawings of a bicycle or unicycle using input features such as wheels, horizontal lines, and vertical lines. We can describe the expectation based on these features. The expectation matrix \( \text{Exp} \) is written below:

\[
\text{Exp} = \begin{bmatrix}
x_1 & x_2 & x_3 & x_4 & \ldots \\
1 & 1 & 0 & 0 & \ldots
\end{bmatrix} \begin{bmatrix}
y_1 \\
y_2
\end{bmatrix}
\]

Expectation matrix \( \text{Exp} \) indicates characteristic bicycles and unicycles based on features \( x_1 = \text{circles (wheels)}, x_2: \) horizontal lines, \( x_3: \) handlebar features, \( x_4: \) seat features. Although binary values are given they can be any real number. For example if 50% of bicycles have seats then the entry can be 0.5.

Two wheels are expected in a bicycle. Horizontal frame and handlebar features are expected for a bicycle. One wheel is expected and no handlebar features are expected for a unicycle. A expectation matrix only requires the expectation of the features relative to the label to be written. This can be determined by a simple averaging function, calculating the co-occurrence of features with labels, or through Hebbian-like learning. It may also be determined by symbolic expressions and language.

Any supervised classifier trained based on the information above, and given \( X = [1, 0, 0.1]^T \), a unicycle, should
respond with $\hat{Y} = [0, 1]^T$, indicating a unicycle label. $X = [2, 1, 1, 1]^T$, bicycle, should generate $Y = [1, 0]^T$.

Optimal feedforward weights are not representative of expectation

Even though supervised weights $W$ may store input-label associations, it is not easy to incorporate expectations into $W$. To demonstrate, let us assume feedforward weights represent expectations and let us make $W_0 = \text{Exp}$. Setting the input to represent a unicycle: $X_{\text{test}} = [1, 0, 0, 1]^T$ and solving $Y = W_0 X_{\text{test}}$ we get $Y = [3, 2]^T$. This is not the expected solution: $Y = [0, 1]^T$. $W$ should be trained using the expectation matrix as a training set.

Solving recognition using $M$

If $M$ represents the fixed points one should be able to make $M$ equal to the expectation matrix ($M_1 = \text{Exp}$), then insert $M_1$ and $X_{\text{test}}$ into either Eq. (10) or (13), and obtain the correct solution. When we do this and wait until the dynamics go to zero $\left(\frac{d\hat{Y}}{dt} \to 0\right)$, the solution obtained for $X_{\text{test}} = [1, 0, 0, 1]^T$ is $\hat{Y} = [0, 1]^T$. Unicycle is correctly recognized. The solution for $X_{\text{test}} = [2, 1, 1, 1]^T$ is $\hat{Y} = [1, 0]^T$ representing bicycle. Both patterns are correctly recognized demonstrating recognition using the expectation matrix.

$M$ vs. $W$

To demonstrate the relation between $M$ and $W$, let us calculate $W_1$ from $M_1$ using the pseudoinverse. Since these matrices may not be square, the standard pseudoinverse method is used where: $W = (M^T M)^{-1} M^T$. $W$ represents the feedforward weights. The transpose $W^T$ is shown.

$$ W_1 = \begin{bmatrix} 0.2 & 0.4 & -0.4 & -0.2 \\ 0.2 & -0.6 & -0.6 & 0.8 \end{bmatrix} \begin{bmatrix} y_1 & \text{Bicycle} \\ y_2 & \text{Unicycle} \end{bmatrix} \quad (26) $$

Compared to $M_1$, $W_1$ is more complex, has negative values, and the values do not clearly indicate fixed-points. To demonstrate that $W_1$ represents correct feedforward weights, let us calculate $Y = W_1 X_{\text{test}}$. Correct answers are obtained: $Y = [1, 0]^T$ for $X_{\text{test}} = [2, 1, 1, 1]^T$ and $Y = [0, 1]^T$ for $X_{\text{test}} = [1, 0, 0, 1]^T$. Correct recognition is obtained either:

(a) with the feedforward-feedback method (e.g. Eq. (13)) using expectation $M$ values from matrix 24 or

(b) with the feedforward method (Eq. (1)) using $W$ values from matrix 25.

The disadvantage of the feedforward method is that $W$ is more difficult to obtain and is sub-symbolic (in Section 'Learning Example' we will also show that it is also more computationally expensive).

Symbolic information

Suppose we want to ask do bicycles have wheels? How many? These are symbolic questions. Using $M_1$ we can look up the label for bicycle, $y_1$, and the feature for wheel, $x_1$, and read the value: 2. If we want to do the same thing for unicycle we can look up the label for unicycle, $y_2$, and the feature for wheel, $x_1$, and read the value: 1. If we attempt this with $W_1$ we do not retrieve symbolically useful information (0.2 and 0.2). Thus $W$ is sub-symbolic while $M$ maintains symbolic access. Symbolic access is possible because $M$ represents expectations.

Learning example

The purpose of this example is to show that generalized learning through $M$ achieves similar results as generalized learning through $W$, but learning through $M$ can be simpler, faster, and more intuitive. Both a single-layer linear perceptron and the feedforward-feedback method are trained on the same learning data and tested on the same testing data. The performance on a PC running Matlab is reported in terms of computation time and number of iterations for both learning and testing. Linearly separable data is used because this is a limitation of single layer networks. We could have used nonlinear data by implementing kernels, but this would not further the goals of simple comparisons since both methods can be used with kernels. 400 Training data points are randomly generated and separated into two labels along a linear separation (see Fig. 3). 100 Separate points are generated as test points. Samples were rejected if they were too close to the separator. Both networks are initialized with random initial conditions. The dynamics, iterations, and computation times may vary. They depend on: whether points close to the linear separator were allowed, the stop threshold for perceptron learning, and the stop threshold for feedforward-feedback iterations. For example, if points close to the linear separator are allowed, learning times increase in the perceptron, and recognition times increase in the feedforward-feedback model (but only for the test cases that are close to the separator).

Performance of the linear perceptron

The perceptron network is trained using the 400 samples until $W$ is found where the number of errors on the training sample is 0. The learning rate is 0.5. A typical data set required about 6000 iterations and about 18 s. Testing the perceptron on the 100 samples was very fast and required about 0.001 s or 0.00001 s per test. The perceptron did
not have any testing errors (100%) as long as the learning and testing points did not fall on the linear separator.

Performance of the feedforward-feedback algorithm

To determine the expectation matrix $M$ from the training data, the points corresponding to each label are averaged. All of the black points above the line and all of the green points below the line are averaged together. Calculating the mean is not iterative and took only 0.02 s. The resulting values from one run are shown in matrix 26.

$$
M = \begin{bmatrix}
    x_1 & x_2 \\
    0.36 & 0.66 \\
    0.69 & 0.33
\end{bmatrix}
$$

(27)

The regulatory feedback Eqs. (12) and (13) were used for testing. The time and number of iterations were sensitive to the threshold value of $dY$ used to stop the simulation. The simulation was stopped when $\sum dY < 0.0001$ and the identification of $Y$ was determined by the node within $Y$ with the highest value. Speed can be selected over accuracy by stopping earlier (e.g. $\sum dY < 0.01$). Accuracy can be increased by stopping later (e.g. $\sum dY < 0.00001$). The time required was about 0.4 s for the 100 tests total or 0.004 s per test.

The average number of iterations per test using this criteria was 21. The performance was analogous to the perception and also did not have any testing errors (100%) as long as the testing points did not fall on the linear separator. When points were close to the separator then errors may occur in both models which may be remedied by adjusting the threshold for learning in the feedforward model or $\sum dY$ in the feedforward-feedback model.

Comparing dynamics in learning and testing

The gradient descent of the perceptron algorithm occurs during learning while the gradient descent of the feedforward-feedback algorithm occurs during testing. Neither gradient descent method (feedforward or feedforward-feedback) was optimized for speed or efficiency. Regardless of model, the gradient descent dynamics took the most time, and consumed the majority of the computational resources. The feedforward-feedback method was faster in learning, about 900 times faster. The perceptron was faster in testing, about 40 times faster. Perceptron learning steps involved choosing one point of data randomly from the 400 training data (in iid form with fixed frequency and random order), evaluating of error, and updating of weights. Thus the dynamics are incorporated over all of the training data.

feedforward-feedback gradient descent was required for each test data point. However, this does not require presenting patterns in iid form like in feedforward learning, so the gradient descent is much faster. The gradient descent mechanism was used to select between the labels (green or black) given the test input and did not iterate over all training data. This saves a lot of time. It took on average 0.004 s per test.

Looking at the combined computational costs for BOTH training and testing, the feedforward-feedback took a total of 0.42 s. The feedforward perceptron method took a total of 18 s and is not symbolic.

Beyond the test and training times, both methods performed similarly and both are governed by the same linear separability limitations. The most important issue in both the linear perceptron and the feedforward-feedback network is to make sure that there are more input features than patterns. Otherwise, the patterns will not be linearly separable. As long as the patterns are linearly separable, when each pattern is presented, the networks correctly identify the patterns. The network can be stacked in a hierarchy and a nonlinear activation function can be used between layers. Moreover, hierarchy can reduce redundancy between the networks, and represent logical-meanings within symbolic networks. However, this is a topic of a follow-up paper. The feedforward-feedback may not be strictly limited by linear separability since the feedforward-feedback implements competition via the feedback connections. For example we were able to approximate an XOR function when we define a node that captures the case when both nodes are active and we also include a bias input. However, this is beyond the scope of this paper.

Multiclass classification

Further testing was done with multiclass classification using randomly generated pattern sets with arbitrary supervised input-label patterns. We tested networks with up to 200 patterns and nodes. As long as the patterns are linearly separable, when a characteristic pattern is presented, the networks correctly identify patterns.

The feedforward-feedback architecture has also been used (Achler, 2011, chap. 4; Achler, Vural, & Amir, 2009) to address mixtures of patterns. In that work, the connections were limited to binary values and single instance training per class but the feedforward-feedback method is the same and obtains similar results. However, in that work, the symbolic nature was not discussed.

Discussion

A major advancement here is that traditionally sub-symbolic information in neural networks is now symbolic in our network that also performs recognition and is now available to symbolic systems. In feedforward algorithms the solutions are opaque and cannot be directly observed through the matrix $W$. Specialized systems that translate information from sub-symbolic neural networks to the symbolic domain seen in certain feedforward models are not required for this model. Thus the feedforward-feedback method eliminates the hypothesized need for separate implicit and explicit systems (Sun, 2002). In the feedforward-feedback network the solutions and purpose can be directly determined from $M$, forming the basis of the neural-symbolic associations. Thus the advantage of using $M$ and feedforward-feedback architectures for the symbolic community is a new ability to model recall directly from neural networks that perform recognition.

The proposed model shares some similarity and properties with Bayesian Networks and methods. Bayesian networks have been invaluable to model cognitive studies,
emulating brain behavior, and modeling the integration of priors. However, they assume a statistical configuration of connections and distributions which is not quite known how to emulate with neurons. To implement Bayesian methods, distributions would have to be explicitly defined and encoded into neurons, which is not trivial. Moreover, exact Bayesian inference is NP-hard. The Bayesian parameter of \( P(X|Y) \) which represents likelihood is similar to \( M \) but also comprises the distributions of the likelihood parameters. Distributions of \( X \) and \( Y \) are needed as well. In the proposed method \( M \) is the equivalent of the average \( \mu(X|Y) \) of the probability without the distribution. In Bayesian networks one multiplication/division is implemented for inference using \( P(Y|X) = P(X|Y)P(X)/P(Y) \). In the proposed model feedforward-feedback signals iterate multiple times until the network settles on an activation. The proposed method is akin to estimating the distributions based on distances between representations.

Bayesian models can display inference similar to "abduction" and explaining away (Pearl, 1988) and so can Generative models such as RBMs (de Penning, d’Avila Garcez, Lamb, & Meyer, 2014; Hinton et al., 2006) and the proposed model. However, the difference between abduction in the proposed model and RBMs is that it occurs during learning in RBM’s, and during recognition in the proposed model. Achler, 2011, chap. 4 focuses on showing examples of how an inductive-like process can select most efficient representation with the least amount of overlap between representations (Achler, 2011, chap. 4).

Similar to the proposed definition of symbolic and weights \( M \), Independent Choice Logic (ICL) (e.g. Poole, 2008) and Hebbian-like learning allows definitions of connections in a context-specific independent manner. Each neuron is defined by the degree of inputs it expects and that is sufficient to define the network. The learning is what is often called “local” and context independent.

Using \( M \), symbolic recognition information can be used to connect models of recognition with models of logic, and emulate fundamental brain functions from recognition processing to logic. For example, brain-motivated cognitive models that utilize symbolic representations to explain how the brain can perform cognitive tasks include: SHRUTI, LIDA, SOAR, ACT-R, CLARION, EPIC, ICARUS (e.g. Franklin & Patterson, 2006; Laird, 2008; Meyer & Kieras, 1997; Shastri, 2000). These methods assume that sensory recognition is processed and symbolically coded representations are available. The processed representations are often hand coded. This is because traditional neural network models are sub-symbolic. Thus these models do not directly incorporate recognition. Without symbolic recognition, this confines many symbolic systems to less-satisfying examples with synthetic starting points. With the neural model presented here, recognition and cognitive models can be better integrated.

Future work is planned expanding on this topic, including hidden nodes, hierarchy and unsupervised learning.

References


& Sun (Eds.), *Hybrid neural symbolic integration lecture notes in artificial intelligence* (pp. 28–45).