THE NEURAL GRAMMAR NETWORK AND ITS APPLICATION TO QUANTITATIVE STRUCTURE-ACTIVITY RELATIONSHIP PROBLEMS

A Thesis

Presented to

The Faculty of Graduate Studies

of

The University of Guelph

by

EDDIE YEE TAK MA

In partial fulfillment of requirements

for the degree of

Master of Science

May, 2009

ABSTRACT

THE NEURAL GRAMMAR NETWORK AND ITS APPLICATION TO QUANTITATIVE STRUCTURE-ACTIVITY RELATIONSHIP PROBLEMS

Eddie Yee Tak Ma
University of Guelph, 2009

Advisor:
Professor Stefan C. Kremer

The Neural Grammar Network is a novel machine learning system that is designed to accept formal language strings as input, and associate each with a real-valued output vector. This is accomplished by combining the syntactic structuring capability of the formal language parser and the general function approximation utility of the heterogeneous dynamic recursive artificial neural network. In this thesis, the NGN is demonstrated as a viable option for the example Quantitative Structure Activity Relationship problem space in computational chemistry. It is shown in this work, that the NGN is capable of acheiving classification accuracy for one QSAR problem within 3% of the leading method, and outperforms all other compared methods in 6 out of 9 regression tasks on average. The NGN fills the niche in machine learning culture wherever syntactic learning of a formal language is too costly or not required for the goal of semantic inference.

ACKNOWLEDGEMENTS

The completion of this work was only possible with the guidance and support of the people I find around me.

I would like to first thank my advisor Dr. Stefan C. Kremer for his expertise and guidance with which this project stayed on track, grew and matured. As well, a great many individuals shared their knowledge, provided timely advice or offered other essential assets. From the University of Guelph, Dr. Deborah A. Stacey, Dr. John Goddard, James Stark, David McCaughan and Tony Thompson. From York University in Ontario, Isaac Ye. From McMaster University in Ontario, Dr. Warren G. Foster. From Xavier University in Louisiana, Dr. Thomas E. Wiese. From eMolecules.com, Craig A. James.

Finally, I want to thank my family and especially my loving *fiancée*, Cara Alicia Tucker who let me stay up late many evenings to work on this project.

Contents

List of Tables

List of Figures

List of Acronyms and Abbreviations

2.5D	2-Dimensional-3-Dimensional Hybrid (Descriptors in QSAR)
ACE	Angiotensin Converting Enzyme (Inhibitors)
AChE	Acetylcholinesterase (Inhibitors)
α (alpha)	Momentum Coefficient
A-MIF	Anchor-Molecular Interaction Fields (QSAR)
Anchor-GRIND	Anchor-GRid INDependent (QSAR)
ANN	Artificial Neural Network
AR	Androgen Receptor (Binders)
BBB	Blood Brain Barrier (Crossing Agents)
BNF	Backus-Naur Form
BPL	Back Propagation Learning
BZR	Benzodiazepine (Receptor Binders)
CoMFA	Comparative Molecular Field Analysis (QSAR)
CoMSIA	Comparative Molecular Similarity Indices Analysis (QSAR)
Cox2	Cyclooxygenase-2 (Inhibitors)
CPP	Confluent Preorder Parser
CV	Cross Validation
DHFR	Dihydrofolate Reductase (Inhibitors)
DRN	Dynamical Recurrent Network
DT	Decision Tree

ER	Estrogen Receptor (Binders)
η (eta)	Training Constant
EVA	Eigenvalue Analysis (QSAR)
FXa	Factor Xa (Inhibitors)
GNF	Greibach Normal Form
GNU	GNU's Not Unix
GPB	Glycogen Phosphorylase B (Inhibitors)
HCP	Hybrid Connectionist Parser
HQSAR	Holographic QSAR
IC_{50}	Half Maximal Inhibitory Concentration
InChI	IUPAC International Chemical Identifier
IUPAC	International Union of Pure and Applied Chemistry
k-NN	k-Nearest Neighboor
LALR(1)	1-Look-Ahead Left-to-Right (Parser)
LDA	Linear Discriminate Analysis
LR	Linear Regression
LSTM	Long Short-Term Memory
MCC	Matthew's Correlation Coefficient
MIF-MIF	Molecular Interaction Fields - Molecular Interaction Fields (QSAR)
MK1	Molecular Kernel 1 (in P-SVM for QSAR)
MK2	Molecular Kernel 2 (in P-SVM for QSAR)
NCTR	National Center for Toxicological Research
	(United States Food and Drug Administration)
NGN	Neural Grammar Network
OrCA	Ordered Child Arborescence
PCA	Principal Component Analysis
PLS	Partial Least Squares
PNN	Probabilistic Neural Network
PRESS	predictive residual sum of squares

P-SVM	Potential-Support Vector Machine
Q	Concordance (or Accuracy)
QSAR	Quantitative Structure-Activity Relationship
r^2_{pred}	Predicted R-Squared
RAAM	Recursive Autoassociative Memory (Network)
RBA	Relative Binding Affinity
RFE	Recursive Feature Elimination
RMSE	Root Mean Squared Error
RNN	Recursive Neural Network
SD	Standard Deviation
SDF	Structure Data File
SE	Sensitivity ($= \text{TP} \div (\text{TP} + \text{FN})$)
SFGA	Spline Fitting with a Genetic Algorithm (QSAR)
SIMCA	Soft Independent Modeling by Class Analogy (QSAR)
SMILES	Simplified Molecular Input Line Entry Specification
SP	Specificity ($= \text{TN} \div (\text{TN} + \text{FP})$)
SPEC	Subsymbolic Parser for Embedded Clauses
SRN	Simple Recurrent Network
SSN	Simple Synchrony Network
SVM	Support Vector Machine
Therm	Thermolysin (Inhibitors)
Thr	Thrombin (Inhibitors)
TN	(Count of) True Negatives
TP	(Count of) True Positives

Chapter 1

Introduction

The application and viability of the Neural Grammar Network (NGN) is demonstrated in this thesis. The NGN is a novel machine learning system that is suitable for formal language string labeling. An example problem space in computational chemistry is approached. Molecules are each expressed as a formal language statement backed by a formal grammar and are then subject to classification and regression toward a chemically, biologically or medically relevant real-value label or vector.

1.1 Artificial Neural Networks

In this work, the term "Artificial Neural Network" should be taken to mean "Back Propagation" or "Gradient Descent" neural network (as there are many other kinds of Neural Networks). We use the term "Artificial Neural Network" for brevity. Artificial Neural Networks (ANNs) (Karray and Silva, 2004) originated as a theoretical mathematical model meant to explain how complex reasoning can arise from simple biological neurons; each in silico neuron originally called a perceptron. In recent times, researchers operate on and with ANNs from a variety of angles. Connectionists continue to maintain the explanatory model building against biology (Sharkey et al., 1994), while computer scientists use ANNs for their practical general function approximation capabilities (Hornik et al., 1989). Indeed

it is this function approximation capacity and the learning algorithms used to drive it that make the ANN an attractive trainable problem solver. The ANN is not immediately good at every task imaginable; but it is good at many tasks, particularly those that can be reduced to vector-to-vector mappings; that is, the emulation of a function that returns some range of elements on a parametrized domain (Haykin, 1998).

1.2 Recurrent and Recursive Networks

The ANN cannot immediately cope with domains that are not expressed as vectors of real numbers, nor is there a straight forward way to encode vectors of variable length that works for every problem space. Vectors of tokens for example must be mapped to unique token vector patterns which are admissible to the ANN and variable length vectors can only be dealt with by exposing the ANN to a fixed amount of input at a time.

Recurrent networks offers a limited solution to sequence sensitivity (Elman, 1990; Hochreiter and Schmidhuber, 1996), by enabling the ANN to cycle some internal representation of the input vector after one round of processing along with new segments of the input vector for a second and subsequent rounds of processing. In this way, the input vector is although spatially segmented, temporally reconstituted in this cyclically processed internal representation. The output is an expression of this reconstitution.

Recursive networks can be mapped so that processing layers of the ANN are mapped literally to the nodes of the graph, enabling the processing of structured data. Fitting the recursive network to structured data is a recent challenge of ANNs (de A. Barreto et al., 2003; Kolen and Kremer, 2001; Kremer, 2001). One method is to use a fixed structure for all possible incoming data. This can work if the data is highly regular and one anticipates a regular partitioning of incoming vectors as in a binary tree. For more complicated tasks, the recursive network may work better if it were restructured for each input vector it is exposed to, capturing the relationship between nodes of the graph.

1.3 Neural Grammar Networks

The Neural Grammar Network (NGN) offers a systematic and regular means to perform such graphical structuring. Given that the input strings are formal statements in a given formal language, the NGN can adapt its shape to reflect shape of the parse tree for each string. This structuring against a regular and formal process ensures that each parse of the same string results in the same structure, and also that each string is parsed against the same formal grammar ensuring consistency in order of precedence. The NGN is thus a recursive neural network that is capable of dealing with input strings of variable length, restructured with each input statement.

1.4 Quantitative Structure-Activity Relationship

Quantitative Structure-Activity Relationship (QSAR) is an interesting problem in computational chemistry. In QSAR, one is interested in mapping some domain of molecules with some range of biological activities (Sutherland et al., 2004). QSAR is interesting because it offers an alternative to costly and time consuming wet lab experiments. The domain of a particular QSAR problem can be defined in any way that allows one to expose all of the salient details required to make a prediction on the range. For the NGN, the domain is the parse tree structure used to encode formal strings describing each molecule. The output can be boolean (classification) for the identification of positive and negative matches for a particular chemical behaviour or a range of real-values (regression) for the identification of a degree of behaviour. QSAR problems are conducted to pre-screen putatively dangerous or illness causing compounds in order to select which compounds should be carried into a phase of testing in the wet lab. This potentially time saving procedure can reduce the labour and equipment costs as well.

1.5 An Opportune Availability of Technology

The NGN arrives as an opportune convergence of technology. It is now that assets in computer science and chemistry are available together. In computer science, formal language parsing and dynamic recursive artificial neural networks form the technology of the NGN. In chemistry, molecular markup languages and an explosion of applicable data sets yield relevance to the development of the NGN.

While the NGN combines formal language parsing with neural networks in a novel manner, it is certainly not the first attempt at neural network (connectionist) parsing and by no means the first attempt at value labeling molecules. These previous attempts and technologies are later discussed along with their relevance to the NGN.

1.6 Thesis Statement

In this thesis, the Neural Grammar Network (NGN) shall be proven as a competitive recent tool in Quantitative Structure-Activity Relationship (QSAR) problems while offering far reduced need for expert knowledge in chemistry. The NGN can be further used as a general machine learning device that operates on domains of formal language strings by encapsulating latent semantic meaning within syntactically formal sequences.

1.7 Roadmap for This Thesis

The next background chapter constitutes an overview of neural network and formal language technologies. This is concluded with a discussion of connectionist parsing approaches demonstrated by others.

Beyond the background chapter, we reveal the implementation and formalism of the NGN. This thesis then visits the target example problem of computational chemistry and

discusses the problem of quantitative structure-activity relationship (QSAR) and the approaches in this domain of other researchers. This leads into the experiments that were performed with the NGN in QSAR, the precise methods and conclusions drawn as well as the overall status of research and development with the NGN.

1.8 Implications

For researchers of neural networks, the NGN offers a device that classifies formal language strings based on the latent semantics derived from syntactic structure while removing the mechanical parsing task from the neural network all together. For researchers interested with QSAR, the NGN offers another alternative graph-based descriptor-free QSAR problem solver; particular problems are better suited for the NGN than any other device as shall be shown in the classification experiments section of this thesis. For all scientists interested in performing classification and regression tasks on domains expressed in formal syntax, the NGN offers a means to do so without any additional expert knowledge in the field.

Chapter 2

Background

This chapter overviews four background areas. These areas are neural network architectures along with common algorithms and math, language parsing concepts, neural network parsing strategies employed by other researchers and finally, other graph based approches in the chosen application area of QSAR. All of these areas are discussed along with their relevance to the Neural Grammar Network (NGN). The neural network topologies visited are the feed forward neural network, the dynamic recurrent network (Elman, 1990), and the recursive neural network (Frasconi et al., 2001). We also briefly visit the random search simple recurrent network (Hochreiter and Schmidhuber, 1996) and discuss its relevance to the NGN.

2.1 Feed Forward Back Propagation Artificial Neural Networks

The term "Artificial Neural Netowrk" (ANN) is used hereafter to mean "Back Propagation" or "Gradient Descent" neural networks. This is done for brevity. The artificial neural network (Haykin, 1998) can be thought of as a function that transforms one real-valued vector to another real-valued vector. It is a universal function approximator (Hornik et al.,

1989). An artificial neural network (ANN) performs this with a so called feed forward pass and can be trained to approximate any function with different algorithms so long as its layers are equipped with a sufficient number of processing nodes. A popular algorithm is gradient descent by back propagation (Rumberlhart et al., 1986; Werbos, 1994). A description of the feed forward pass and back propagation will be covered after discussing the way data is bound to the ANN.

Structurally, the ANN consists of one or several layers of real-valued two-dimensional matrices called weights, and one-dimensional real-valued vectors called activations. An optional column matrix of biases can be used as well to change the amount of activation a unit produces in response to input by either predisposing it to a greater or reduced activation threshold. Each column matrix is conceptually associated with each two-dimensional weight matrix so that weights and biases are referred to simply as weights for brevity. Layers of activations and weights are arranged alternatively, so that the activations of the final layer of the network are the overall output of the network. Inputs are fed into the first layer of weights in the same way activations are fed upward through the network. Processing a vector of inputs or activations into a subsequent vector of activations consists of a mathematical transformation taken along with the corresponding weights.

For clarification, the notion of a 'unit' refers to an index used to reference a single value in a vector. In a 2D matrix, the notion of a unit refers to a row of values. This mapping of single matrix rows to single vector values is useful in math used for neural networks.

One can mathematically formalize the objects that a neural network consists of as follows.

$$N = \{A, W\}$$

Whereas a neural network N is a two-tuple of activations A and weights W.

$$A = [\vec{a}_{\text{input}} \ldots \vec{a}_{\text{output}}]$$

The set of activations A is a stack of vectors $[\vec{a}]$. Each of these activation vectors is also called a node layer. The input node layer $\vec{a}_{\text{input}}$ may be separated from the output node layer $\vec{a}_{\text{output}}$ by an arbitrary number of hidden node layers. Each of these kinds of layers can be called simply input layers, hidden layers and output layers respectively. To clarify how different layers relate to each other, a layer that sends data upward to the layer above will be called a child layer and the layer that receives data from the layer below will be called a parent layer.

$$W = \{[w_{(\text{input}\rightarrow\text{input + 1})} \cdots w_{(\text{output - 1}\rightarrow\text{output})}], [\vec{b}_{\text{input + 1}} \cdots \vec{b}_{\text{output}}]\}$$

The weights W are a two-tuple consisting of a stack of two dimensional matrices $[w]$, and a stack of vectors $[\vec{b}]$. The matrices and vectors are called weight matrices and biases respectively. The values entered into the weights and biases are generally small random values centered about 0.0 (e.g. $[-0.3, +0.3]$); these values are used to break system bias. Weight matrices abstractly connect node layers together so that the number of rows is the same as the number of units in the parent node layer and the number of columns is the same as the number of units in the child node layer. Weight layers are identified in this paper with the notation $w_{(\text{child}\rightarrow\text{parent})}$. Only parents have bias vectors. These are special values which will be shown to influence the activation of a given node layer.

The structure of an artificial neural network is shown abstractly in Figure 2.1. This example neural network has three layers; one input layer, one hidden layer and one output layer. The input layer has four activation units, the hidden layer two and output layer four.

In Figure 2.1, the 'Classic View' on the right column is a common representation used in other texts. The 'Fat Arrow View' is adopted by this paper as it is less cluttered. In this view, rectangles are node layers that enclose processing units shown as triangles (for input layers) and circles (for hidden layers and output layers). Within the triangle or circle icons, an integer indicates the number of input or other processing units found in that layer.

Thin arrows are individual entry values (units) in a given vector. The weights and biases bridging a child layer to its parent are represented as an arrangement of line segments in 'Classic View' and as fat arrows in 'Fat Arrow View'.

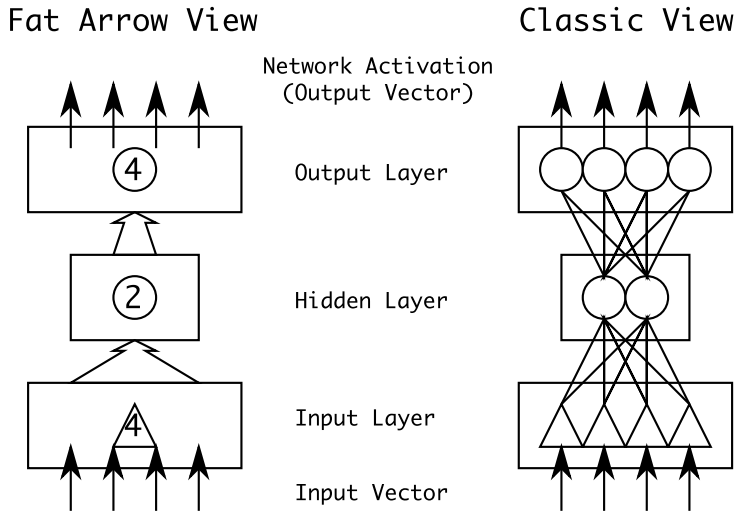


Figure 2.1: A feed forward artificial neural network with one hidden layer.

2.1.1 Feed Forward Pass

A feed forward pass consists of traversing the stack of neural network layers starting at the inputs and applying a series of mathematical transformations.

The product is taken of the input vector of the input layer $\vec{a}_v$ against the weight matrix bridging the input layer u to the first hidden layer v so that the weight matrix is designated $w_{(u \to v)}$. The resulting product is added to the bias vector $\vec{b}_u$ resulting in a new vector $\vec{net}_u$.

$$\vec{net}_u = w_{(u \to v)} \times \vec{a}_v + \vec{b}_u$$

For many networks, each unit of this vector is then subjected to a transfer function (often the logistic function) which rescales the the output range to a new range that is admissible to the next neural layer. The vector $\vec{a}_u$ is the result of the logistic function transformation

on the vector $\vec{net}_u$ shown below. It is the vector $\vec{a}_u$ that is used as the final activation of this layer.

$$\textbf{Logistic}(\vec{net}_u) = \frac{1}{1 + e^{-net_u^{(i)}}}$$

The transformed output vector of the input layer and first weight matrix and bias is the activation of the first hidden layer. This process is then repeated so that the activation of the first hidden layer is used with the next weight matrix and bias to calculate the activation of that hidden layer's parent.

The transfer function that is used to scale the output determines the useful range of values that can be used with the network during training. The logistic function which is used with the NGN is asymptotic against $(0.0^+, 1.0^-)$. Other transfer functions include tanh $((-1.0)^+, (+1.0)^-)$ and the signum function which returns a discrete -1.0 or $+1.0$ thresholded against 0.0.

The final output of the neural network is the activation vector of the output layer.

2.1.2 Gradient Descent with Back Propagation

A popular method of training the ANN is gradient descent. Gradient descent entails iteratively refining the values of the weight matrices and bias column vector so that the output vector produced is ever closer to a vector considered correct corresponding to the input vector. Back propagation is an algorithm that does this with three passes of data. A feed forward pass as explained above first yields an output vector followed by a gradient finding pass and a weight update pass.

We continue with an explanation of finding the gradient of error. Note that in the following, subscripts define which layer is referenced while a superscript defines an iteration over the units of that layer.

This network output $\vec{a}_v$ is compared with what is considered the correct output $\vec{a}_T$, and blame for erroneous values is assigned to the weights leading to the output activations.

Feeding the blame for erroneous values is repeated given each pair of activations and its associated weights. This blame (or gradient of error) is represented by a vector $\vec{\delta}_v$.

The following equation calculates $\vec{\delta}_v$ for output layers.

$$\vec{\delta}_v^{(j)} = \left[\left(\vec{a}_v^{(j)} \right) \left(1 - \vec{a}_v^{(j)} \right) \left(\vec{a}_T^{(j)} - \vec{a}_v^{(j)} \right) \right]$$

The blame is calculated in turn from output layer back to input layer as errors are calculated with respect to the error of a parent layer. For a hidden layer or an input layer, the blame of its parent $\vec{\delta}_u$ along with the weights connecting this layer to its parent layer $w_{(u \to v)}$ and this layer's activations $\vec{a}_v$ are used in the computation of its own blame $\vec{\delta}_v$, where u represents this layer and v represents its parent layer.

The following equation calculates $\vec{\delta}_v$ for hidden layers and input layers.

$$\vec{\delta}_v^{(j)} = \left[\left(\vec{a}_v^{(j)} \right) \left(1 - \vec{a}_v^{(j)} \right) \left(\sum_{i=1}^{|u|} \vec{\delta}_u^{(i)} w_{(u \to v)}^{(ij)} \right) \right]$$

This feed backward blame assignment pass constitutes the second pass of data.

The final pass of data constitutes a calculated change required for each value of each weight matrix and column bias vector. This is done as a feed forward pass starting with the weight matrix connecting to the input layer.

Calculating the weight changes $\Delta w_{(u \to v)}$ and bias changes $\Delta \vec{b}_u$ follows the next two mathematical functions respectively where u is a given layer, v is the child of that layer, $\vec{a}_v$ is the activation of the child layer and $\vec{\delta}_u$ is the blame for error assigned in the previous step for this layer.

Two equation parametres remain constant throughout training, they are η, the training constant and α, the momentum coefficient. Both of these parametres are arbitrarily deter-

mined by the user of the algorithm. In general, each of these terms can be set to [0.1, 0.9]. The training constant abstractly corresponds to the amplitude by which a network reacts to incorrect outputs. The momentum coefficient defines how much to adjust a network based on previous adjustments.

$$\Delta w^{(ij)}_{(u\to v)}(t) = \eta \vec{\delta}^{(i)}_u \vec{a}^{(j)}_v(t) + \alpha \Delta w^{(ij)}_{(u\to v)}(t-1)$$

Notice the use of the parameter t to indicate the time step of training. In the above and below equations, $t-1$ indicates a reference to the most recent weight matrix or bias vector respectively and is used in the momentum term. This momentum term is important in preventing the network output from oscillating about an intended target.

$$\Delta \vec{b}^{(i)}_u(t) = \eta \vec{\delta}^{(i)}_u(t) + \alpha \Delta \vec{b}^{(i)}_u(t-1)$$

After the weight changes are calculated, they are simply added to the weights and biases of the network.

The three steps are repeated on the network until an acceptable threshold for error is accommodated. At this point, the network is said to be converged. Different measures for error or accuracy can be used depending on the problem type.

From the feed forward neural network, some modifications in the flow of data can be made which result in the following recurrent architectures.

2.2 Dynamical Recurrent Networks

A dynamical recurrent network (DRN) (de A. Barreto et al., 2003; Kolen and Kremer, 2001; Kremer, 2001) is an artificial neural network such that a vector of activation is not only fed

forward onto the next weight matrix as above, but could also be fed onto another weight matrix that exists earlier in the ANN. In this way, the data can be thought of as having passed recurrently spatially and temporally. Recurrence allows a network to combine and process past data that has already been encountered with new data. DRNs offer a solution to processing data of variable length by iteratively combining together a sequence of data. The history of data seen by a network while recognizing a single vector is stored as a the state of activations within the network.

The structure of an example DRN is shown in Figure 2.2 using the 'Fat Arrow View' established in Figure 2.1. Notice that activation taken from the hidden layer is recycled and readmitted to the input layer so that it can be combined with incoming data. This allows the network to operate on temporal relationships within the data.

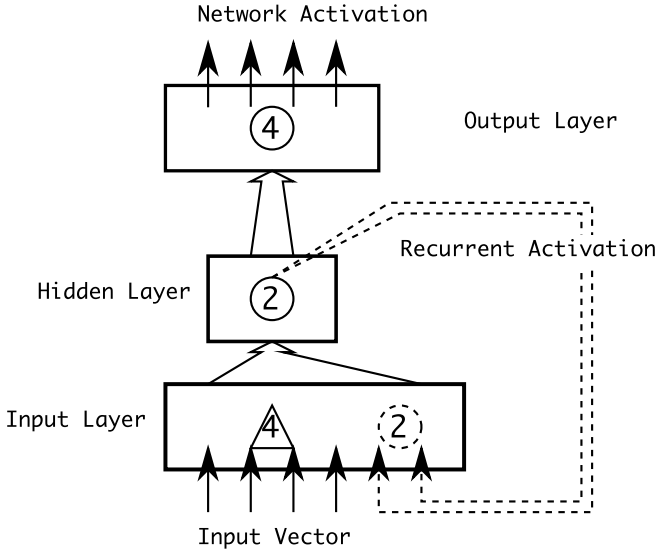


Figure 2.2: A dynamic recurrent network (Elman, 1990).

The same DRN as in Figure 2.2 is unrolled over three time steps in Figure 2.3. Notice that the total length of the input vector is twelve units, but the input layer is only four units wide. The recurrent network iteratively accepts four units worth of input at each timestep until the entire vector is consumed. Interim incomplete network outputs are generated before a completed output vector is created. These incomplete values represent the return of the function approximated by the network given the first four and eight units of input respectively. Only the completed output returns the intended result of the function as it

operates on all twelve input values.

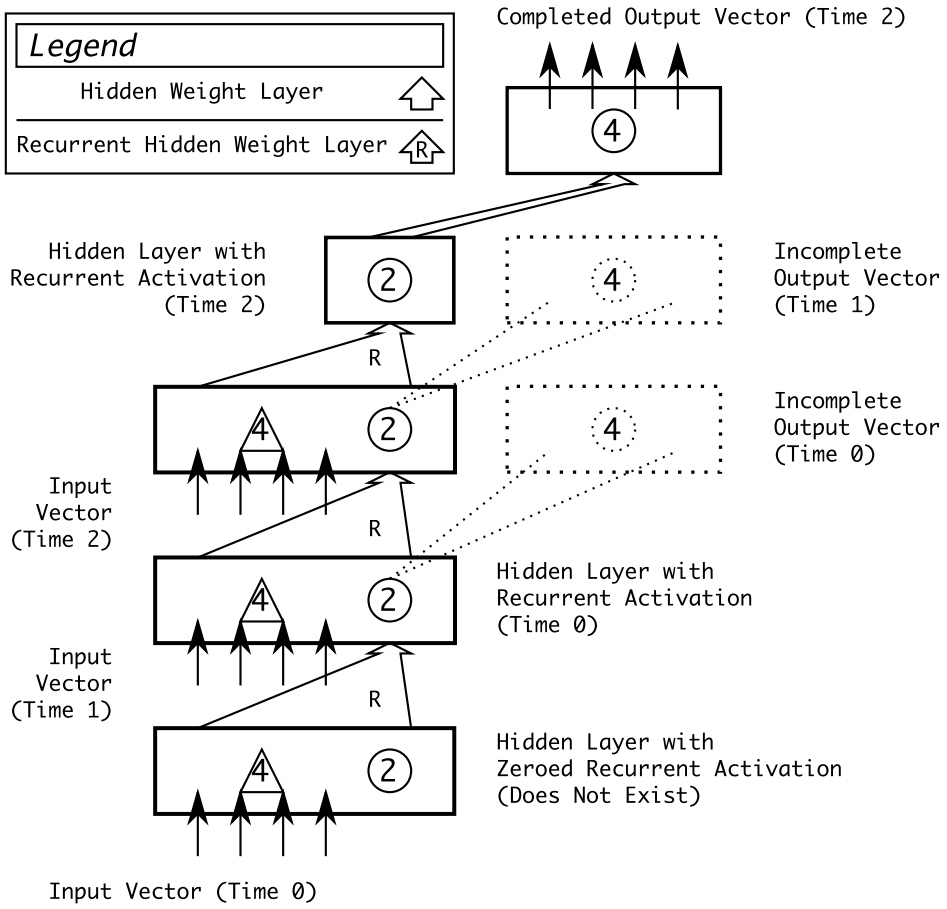


Figure 2.3: A dynamic recurrent network unrolled over three time steps.

2.3 Recursive Neural Networks

Recursive neural networks (RNN) (Frasconi et al., 2001) are similar to recurrent artificial neural networks in that they both combine past data together with present data. How this is accomplished is different. In a DRN, activations from parent layers (and other layers above a given layer) are copied verbatim to a given layer; whereas in a RNN, a reference for a given weight matrix may occur more than once in a network allowing that single matrix more than one opportunity at processing the data. The term recursive is used since each shared weight matrix may be updated by more than one state of data at each time step of training. A single weight matrix and set of biases may even process different vectors simultaneously.

The topology of a RNN can be mapped to a directed acyclic graph. Conceptually, this allows for more precise control over when and how to repeat data processing. The neural grammar network benefits from the use of this additional data processing control.

An example of a recursive ANN is depicted in Figure 2.4; this example illustrates the ability for a single weight layer to process several input vectors simultaneously. Notice that multiply referenced weight layers are labeled with the same letter. This recursive shared referencing distributes processing of input over time and space. In this example, this distribution may be useful for compressing the information in the leaves of a binary tree.

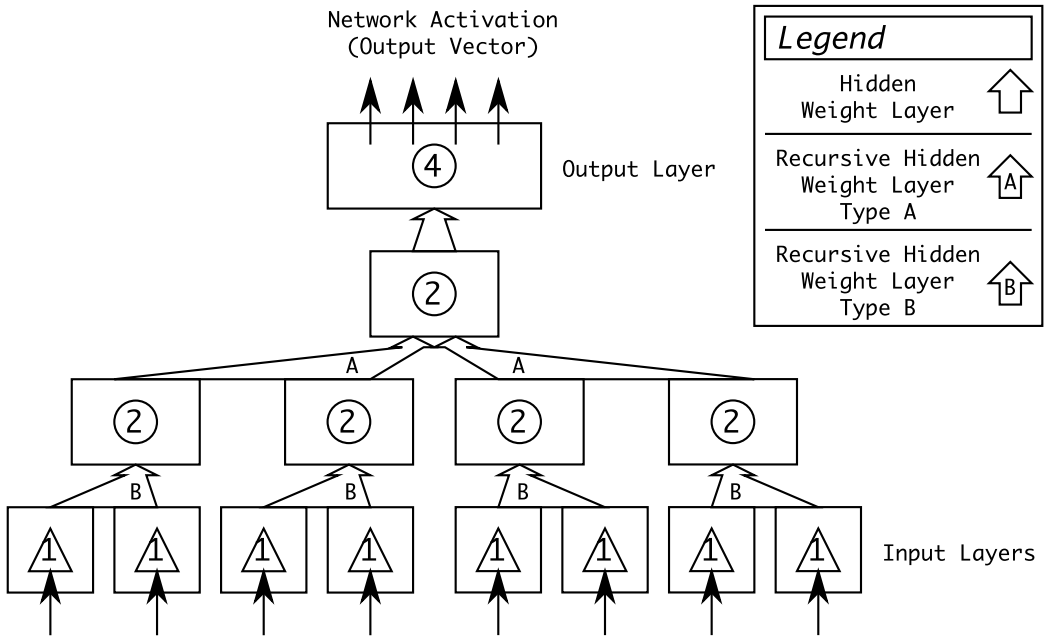


Figure 2.4: A recursive neural network.

2.4 Homogenous and Heterogeneous Recursive Neural Networks

A recursive neural network may be composed of the same weight layers throughout the network (homogenous) or it may be composed of different weight layers (heterogeneous as shown in Figure 2.4). The NGN is one such heterogeneous recursive network. The

assignment of which layers are selected at what height of processing in the tree permits a greater specificity to the system. This delegation of fractional processing is important as it reduces the amount that each kind of weight layer is required to learn. This additional specificity allows each layer of processing to become particularly good at capturing its own niche of salient information.

An outstanding challenge in the design of heterogeneous architectures is how best to decide the shape of the network. If a network is too short and input vectors are received at different heights of the network, a layer needed to break linear inseparability may be missing. If the network is too tall, and there are too many redundant layers, the back propagation of error may not sufficiently correct the network as these values decay exponentially as they are transformed along with more data at each node layer in back propagation.

2.5 Random Search Simple Recurrent Networks

A final innovation in neural network design to discuss is the random search simple recurrent network (or random search net) (Hochreiter and Schmidhuber, 1996). While the topology is not immediately relevant to the way the NGN is structured, the way one trains the random search net is relevant. A random search net is a small network, consisting of few nodes and sparsely connected weight matrices. One way to sparsely connect a weight matrix is to zero out values that represent connections that have been mathematically cut away, never changing those values from zero. A random search network is trained by randomly guessing the final values of the weights. A network that does not satisfactorily generate a target output vector is thrown away and a new one instantiated in its place with new random weights and biases. Keeping these networks small and sparsely connected allows one to reduce the size of the search space by limiting the number of parametres (or values) seen by the system. An example of the Random Search Recurrent Network is shown in Figure 2.5, where a sequence of bits are presented to the network over time (shown as input value arrows in a queue). Based on the entire history of presented values, this particular example successfully produces the values associated with the binary exclusion problem. In

this example, the output layer receives activation from all other units in the net, each hidden unit receives activation from itself (shown as the circular arrows), the input layer (the one input unit) and the output layer (the one output unit).

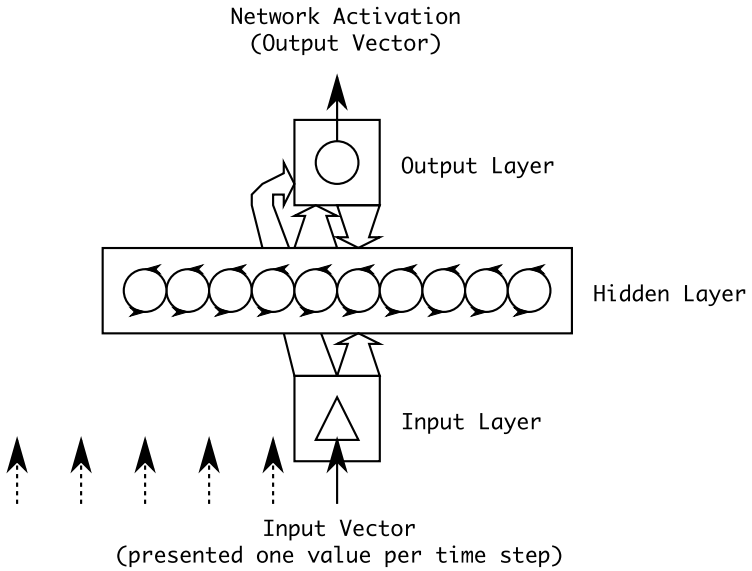


Figure 2.5: A random search simple recurrent network (Hochreiter and Schmidhuber, 1996). This network is successful in solving the bit parity problem.

Random searching is the inspiration for the large 'blindspot' weight initialization of the NGN discussed later.

In the preceding sections, several neural network architectures were discussed and explained for their relevance leading up to the topology of the NGN. In the next sections, a similar treatment is provided with respect to the fundamentals and approaches used in language parsing.

2.6 Background Language Parsing Concepts

In the following overview, language parsing concepts which are relevant to the NGN are discussed. Language parsing is discussed briefly to set up concepts needed later on in this work.

Parsing is the task of deriving meaning from a sequence of statements so that a sequence

of corresponding functions are executed. The notion of syntax refers to the legal grammar and consistent logical representation of a language while semantics refers to the meaning that is derived from these legal statements. Parsing can be thought of as the recognition and translation of syntactically correct statements into corresponding semantics.

For the NGN, syntactic structuring of a string is performed by an internal flex (flex, 2008) lexicological analyzer and GNU bison (GNU bison, 2008) parser while the neural network components deal with the semantics. This is discussed in detail in the next chapter on implementation details.

Two kinds of languages exist; they are formal languages and natural languages. Formal languages are mathematically precise. In practice, programming languages, markup languages and mathematical forms are all formal languages. Natural languages are those that are used in day to day conversations by humans. Ambiguities exist in a language when there is more than one possible interpretation of its syntax. Formal languages are carefully designed to avoid ambiguities while natural languages contain many.

While formal languages can be parsed with deterministic parsers, clever strategies must be employed to deal with the ambiguities of natural languages. It is due to the nondeterministic nature of natural languages that adaptive machine learning approaches (including connectionist approaches) have been developed.

For the purposes of this background, only formal language parsing as performed by deterministic parsers is discussed. The discussion is also limited to the syntactic aspect of parsing, while semantic representation is discussed in later sections. An example formal grammar in Backus-Naur Form (BNF) and a subset of its corresponding language are shown here to demonstrate the notion of formal languages. The grammar described has a start symbol S, two kinds of internal symbols A and B, with the tokens c and d. The symbol '$\rightarrow$' indicates that the left hand symbol expands to the right hand statement (dubbed the production operator). The symbol '|' is a logical 'or' indicating that the statements on either side are possible productions.

A BNF Grammar G for language L:

$$S \rightarrow A|cA$$

$$A \rightarrow B|BA|Ae$$

$$B \rightarrow d$$

The above grammar contains a recursive statement $A \rightarrow BA$ which allows for recursive expansion and thus strings of variable length. Illegal statements in a language are those that cannot be parsed given a formal grammar for the language. Below are some strings that can match the above formal grammar and are thus legal.

A few example legal statements in language L:

$$\{d, de, cdd, cdde, ddde, ...\}$$

The valid parses for these strings is presented as parse trees in Figure 2.6.

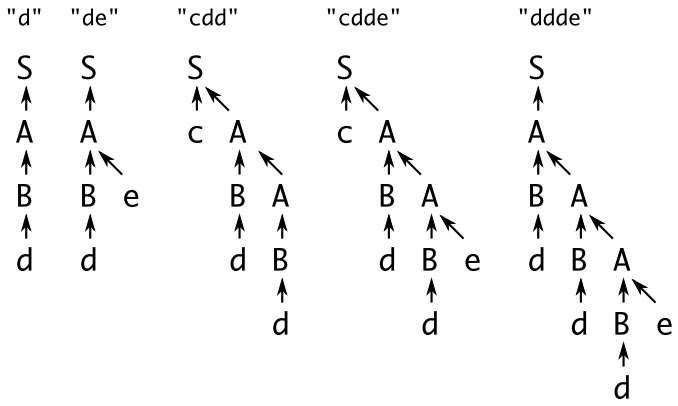


Figure 2.6: A set of parse trees for the example strings parsed against the grammar in Backus-Naur Form presented earlier.

This concludes the background required for understanding language parsing as it relates to the NGN in this work. We now turn our attention to neural network (or connectionist) parsers employed by other researchers.

2.7 Neural Network Parsing Strategies

Neural network parsing strategies and their relevance to the NGN are discussed here. The NGN uses ANN components to derive semantics in order to perform pattern recognition, while tasking an internal parser to perform syntactic structuring. Other neural network parsing strategies can operate on either the semantic or syntactic task, or both and may be designed to derive such regularities in both natural languages and formal languages. As will be revealed later, such connectionist parsers may also perform case-role analyses which is similar to syntactic structuring but with the knowledge of each token's intended semantic role. It should be noted that connectionist parsers are inherently different than the NGN in that most are developed with the intention of operating on natural languages rather than formal ones. This discussion reveals the relevant concepts and insights between the two. In additional, some claims are made about connectionist parsers' architecture or behaviour and their similarity to theoretical procedures performed by the human brain. The NGN strives to make no such claim and is designed to focus on recognition of the structured syntax of formal languages only.

It should be noted that in this thesis, the term 'neural network' may be used instead of 'connectionist' and the term 'statement' instead of 'sentence' to mean a complete line of input units, where each such unit is referred to as a 'token' instead of a 'word'. This is done in order to keep discussion simple and consistent.

There are two general kinds of neural network parsing strategies, they are the so called holistic (Lane and Henderson, 2003) and incremental parsers. Incremental parsers construct a parse tree as each token of input is presented, while holistic parsers extract a parse tree from an internal representation that buffers and compresses an entire statement.

2.8 Incremental Neural Network Parsers

Example incremental parser approaches include the Hebbian Parser (Hadley and Hayward, 1997), the Simple Synchrony Network (Lane and Henderson, 2003) and the Hybrid Connectionist Parser (Kemke, 2002).

The Hebbian Parser accomplishes parsing as follows. Words are presented as tokens in an input layer, and are connected to a semantic layer which can be treated as the network output. The so called semantic layer organizes to reflect the syntax of the input sentence. The output layer is actually a tree consisting of a Master Proposition Node, and many Modifier Proposition Nodes (where Proposition Node is shortened to pNode). A master pNode connects to three sites which bind to input activations. These sites correspond to tokens that are learned to be one of 'agent', 'action' or 'patient' words (tokens). Modifier pNodes have an additional site 'modifier', which connects to the 'agent' or 'patient' of other pNodes for tokens that have already been parsed. Syntactic labeling is accomplished, and the parse tree is derived from the input sentence by performing such proposition-wise attachments until the sentence tokens are exhausted. This parsing model utilizes a simple recursive topology by identifying previous propositions that are to be treated as elements ('agents', 'patients') of a higher proposition. A schematic of the Hebbian Parser is shown in Figure 2.7.

Subfigure 2.7 A (left half) illustrates how data is input to the Hebbian parser. Input vector patterns are transformed and activate pNodes in the semantic layer above. A single master pNode commences parsing these tokens by assigning them case roles. The modifier pNodes then nests previously parsed tokens along with new tokens causing the semantic layer to accommodate more complicated statements. Subfigure 2.7 B (top right) is a schematic of a so called modifier pNode whereas a master pNode is lacking a 'modifier' site. Subfigure 2.7 C (right edge) is an example parse of the sentence 'Jane sees Bill who likes Mary'. Subfigure 2.7 D (bottom right) is an example parse of the sentence 'Bill who likes Mary sees Jane'.

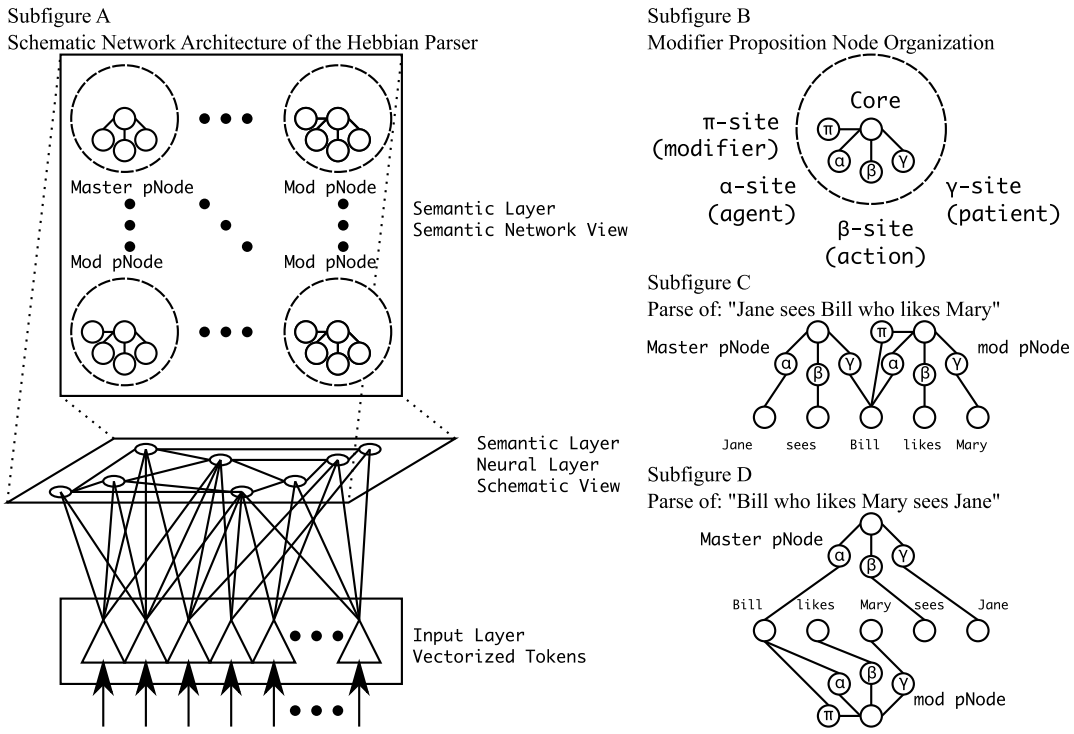


Figure 2.7: A schematic of the topology of the Hebbian parser.

The Hebbian parser is suitable for the case labeling and recursive structuring of clauses in natural languages. With respect to the problem chosen for the NGN, the Hebbian parser could be used to represent molecules after modification to the case role labels it defines. For instance, the noun case tokens for 'agent' and 'patient' could be replaced by tokens that represent atoms or functional chemical groups while the verb case token 'action' could be mapped to different kinds of chemical bonds. Aside from merely representing the molecules however, it is unlikely that the Hebbian parser can be used to learn and discriminate against latent semantic features as its design deals mostly with resolving syntactical structure.

A Simple Synchrony Network (Lane and Henderson, 1998) (SSN) does the same sort of case role labeling in syntax construction, but with emphasis on temporal sensitivity so that a given token is classified based on the time step that it was admitted. Inputs are analyzed with two hidden layers simultaneously so that each token is analyzed for its present syntactic role and compared to the state of syntax of recent tokens. The final parsed representation is a deep recursive tree of output units. The SSN is a training capable variant of the Temporal Synchrony Variable Binding (Shastri and Ajjanagadde, 1993) network. A schematic of the

SSN is shown in Figure 2.8.

The node layers in Figure 2.8 associated with Entity Processing (shown in the left half) are addressed first. These pulsing layers are here represented as a stack of subtrees. Depending on the case role which the SSN is expecting in a particular timestep, a specific subtree from the stack associated with that case role is selected to take input. The node layers associated with Gestalt Processing (shown in the right half) are said to be non-pulsing as they always receive input.

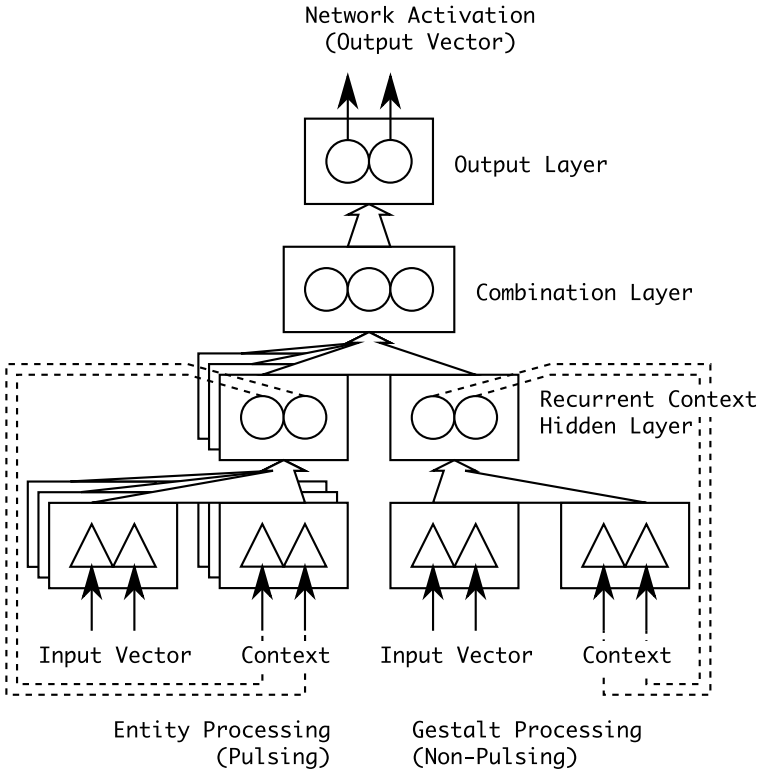


Figure 2.8: A schematic of the Simple Synchrony Network.

The SSN is similar to the NGN in that the processing of specific kinds of tokens (entities) of an input statement are left to specific kinds of layers related to each corresponding kind of information. The SSN accomplishes this differently than the NGN however because it breaks data apart with a recurrent architecture with cycling activations whereas the NGN breaks data apart using a recursive architecture with spatial arrangements of specific processing layers. Like the Hebbian parser above, the SSN could be made to process molecules like the NGN by changing the kinds of entities it processes to those related to the graph

representation of molecules.

Finally, Hybrid Connectionist Parsers (HCP) are discussed. This is an example of a localist architecture (Palmer-Brown et al., 2002). Localist architectures strengthen weights between neural network units which capture the syntax most correctly based on supervised learning while weakening connections that are less likely to be correct. By contrast, a distributed architecture offers no explicit inhibitory mechanism (e.g. a simple feed forward network can be said to be a distributed architecture). For the HCP, the correctness is inferred from the regularity and predictability of performance. Neural network units that correspond to hypothetical grammatical elements are pieced together along with input tokens as each token is presented. The parse tree is constructed this way. Multiple subtrees for the same statement resulting from many hypothetical parses all contribute to this tree structure. These parses are all linked together so that hypothetical internal grammar elements serve as intermediate hidden nodes. Parses that are deemed correct have their corresponding weights strengthened and inhibit competing weights. In the HCP, the final parse tree is represented by the literal mapping of input units and hidden units and the weights between them. A schematic of the HCP is seen in Figure 2.9.

In Figure 2.9, the parallel construction of multiple subtrees is shown. Successive merging of subtrees that share parents results in one final parse tree. Hypothesis nodes indicating alternative hypotheses corresponding to different productions are not shown in this diagram. Only the successful productions are indicated. Horizontal dotted lines indicate a match between parent nodes suitable for merging subtrees.

The HCP is the most structurally similar system to the NGN out of all of the incremental parsers described above. The HCP in its present form is designed to deal purely with the syntactical structure of a statement and does not accommodate any semantic information making it a poor immediate choice for a molecular classifier. Compared with the Hebbian parser however, the HCP is a more likely candidate to be altered to work with latent semantic data. Explicitly defining nodes in each layer of the HCP to operate on hypothetical syntactic derivations while allocating other nodes in that layer for the molecule classification

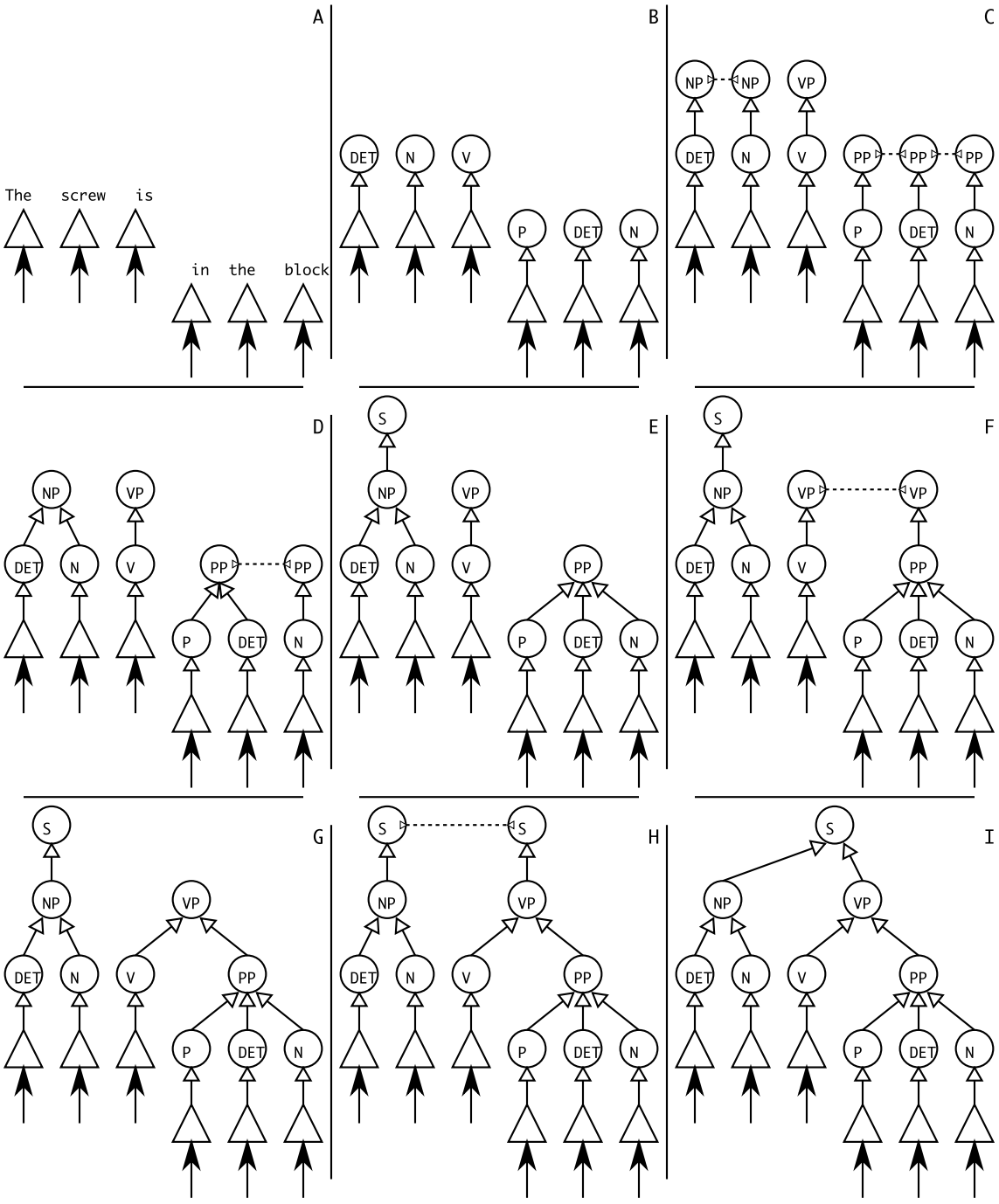


Figure 2.9: A schematic of a Hybrid Connectionist Parser constructing a parse tree for the sentence "The screw is in the block".

task or other semantic task is the most straight forward alteration. This is in contrast with the Hebbian parser which would require an in depth understanding of the best hypothetical decomposition of a molecule (or other task) into notions relating to case roles.

2.9 Holistic Neural Network Parsers

Several holistic neural network parsing topologies will be discussed here, they are the general Feed Forward Artificial Neural Network, the Simple Recurrent Network, the Recursive Autoassociation Memory Network (Pollack, 1990), the Subsymbolic Parser for Embedded Clauses (Miikkulainen, 1996) and the Confluent Preorder Parser (Shiu et al., 1996).

The Feed Forward Artificial Neural Network has been used as an engine to both parse a statement and execute its corresponding semantics. The internal state of activations abstractly represents the parse of a statement. The statement is read in a sliding reading frame (also called a window in time). This abstract representation is never manifest to the observer as only the semantics of the statement are expressed as output. The sliding reading frame encompasses a fixed number of tokens at a time, and at each time step processes one less token to the left and one more token to the right. Pattern association is therefore non-temporal and all relevant network activations must be deduced by the present frame only. The feed forward network is thus used exclusively for transcoding. An excellent example of this transcoding behaviour is the early work done to emulate the process of reading text aloud by a project known as NETtalk (Sejnowski, 1988).

Simple Recurrent Networks (SRNs) perform this task while committing a variable number of tokens to the internal representation simultaneously by admitting a certain number of tokens at a time. The internal recurrent activations are updated at each time step with these new tokens. The internal pattern of activations is thus a representation of all inputs seen by the system for a particular statement. The SRN is thus capable of both transcoding and producing a final output.

Recursive Autoassociation Memory (RAAM) networks have been used for parsing. Three node layers comprise the RAAM; an input layer, a hidden layer and an output layer. The input layer can be considered two discrete vectors of the same size. The output layer is also two vectors of the same size. The RAAM is designed and trained so that one of the two input layer segments is mapped to receiving new input tokens while the other receives recurrent activation from the hidden layer; one output layer segment is trained to echo the input seen, while the other is trained to echo the activations of the hidden layer. Conceptually, the RAAM can be considered an encoding-decoding duplet, so that the weights between the input layer segments and the hidden layer compose an encoder while the weights between the hidden layer and the output layer segments compose a decoder. For parsing, the activations of the output layer are used directly as a representation of the stack of inputs seen by the network. A schematic of a RAAM is shown in Figure 2.10.

The network in Figure 2.10 consists of three node layers. There are four weight layers that can be divided two ways. In the above schematic, a horizontal split divides the weight layers so that the top two are decoders while the bottom two are encoders. A vertical split divides the weight layers so that the left half operates on the activations of the hidden layer and the right half operates on the activations of the input layer.

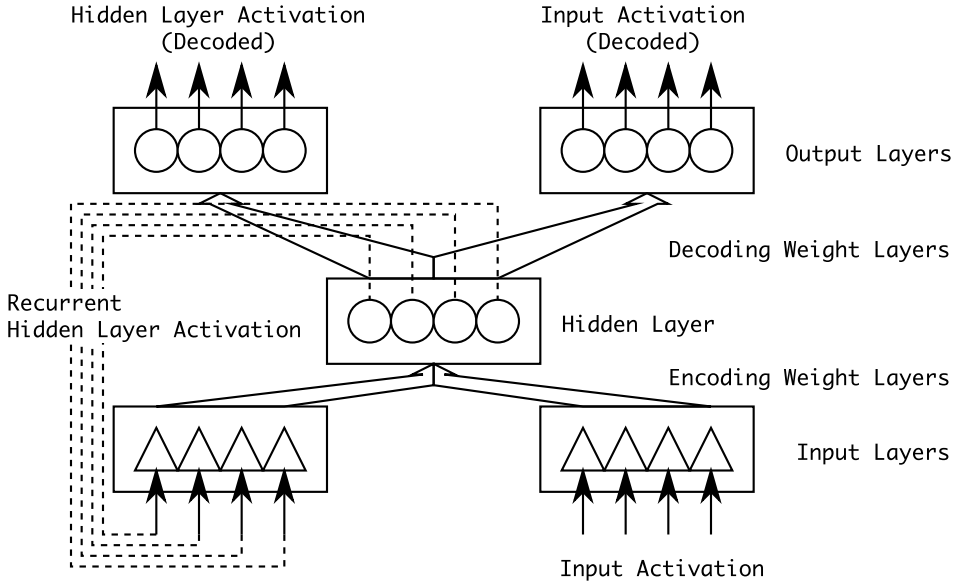


Figure 2.10: A schematic of a recursive autoassociation memory (RAAM) network.

Interestingly, all three of these architectures have been used in tandem. For the Subsymbolic Parser for Embedded Clauses (SPEC), a feed-forward network first accepts tokens and labels them for their syntactic role. The SRN then breaks down statements into smaller segments. If the result of these two steps yields a a completed statement, an agent-action-patient triplet is output similar to the semantic mapping given by the Hebbian parser. If there is insufficient data, the segment is further treated. The RAAM accepts these segments and acts as a stack that will retrieve these incomplete segments related to both the order of admission as well as other complete triplets seen in the past. The SPEC is shown schematically in Figure 2.11.

Where necessary, abbreviations are used in Figure 2.11 as follows. 'Parser Previous' and 'Parser Prev' are copies of the Parser Hidden layer in the previous time step. 'Parser Hid' is short for Parser Hidden layer. 'Stack Rep' and 'Stack Represent' are short for 'Stack Representation', a state buffered in the RAAM.

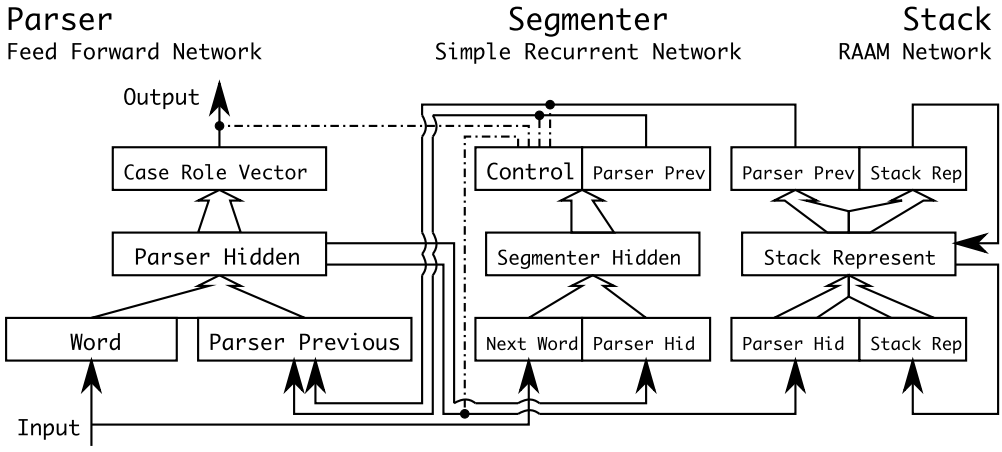


Figure 2.11: A schematic of the Subsymbolic Parser for Embedded Clauses (SPEC).

Finally, the Confluent Preorder Parser (CPP) is overviewed next as the final holistic parser discussed in this paper.

The CPP is architecturally the same as two RAAM networks so that the middle hidden layer is shared between. The inputs and recurrent activations of both RAAMs are fed to the same hidden layer with different encoders which then feeds out to the output layers of the two networks with different decoders. The CPP is trained by exposing the network to a

string of tokens in a sentence simultaneously with the preorder traversal of its correct parse tree. This method of training abstractly induces the identification of the correct parse tree traversal against each string of tokens. The CPP is shown as a schematic in Figure 2.12.

Notice the general design of the RAAM in Figure 2.12 featuring the encoding-decoding weight layer duplet and recurrent feedback from select hidden layer activations to their respective input layers.

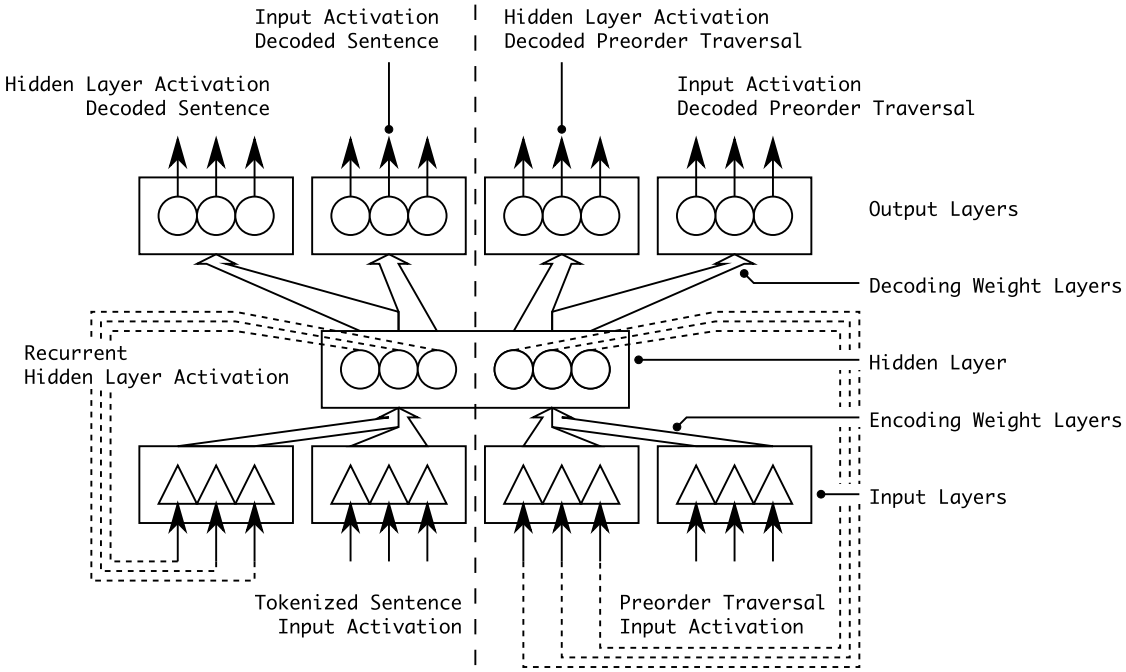


Figure 2.12: A schematic of the confluent preorder parser.

The SPEC and CPP are both architectures that are vastly different than the NGN. Similar to the Hebbian parser and HCP above, both of these devices are primarily tasked with the construction of the parse tree (by case role or traversal respectively). There is no obvious way to insert additional semantic data into the SPEC, as even its output has its own well defined three-tuple grammar. The CPP however appears to be able to accommodate additional semantic data. The example molecule recognition task explored with the NGN might be accommodated by the CPP by simply adding a third set of vectors to the RAAM architecture that deals with molecular classification or regression values. The use of a mechanical parser would be needed prior to the training phase of the CPP to generate the preorder traversal however.

2.10 Discussion for Neural Network Parsers

An additional enhancement that has been explored for connectionist parser approaches is the use of a large body (corpus) of example labeled statements from which to draw syntax fragments (Tepper et al., 2002). Such a corpus of examples could be thought of as a means to encapsulate a large natural grammar by example as a concise formal definition is not possible.

From the above discourse, one can say that the NGN is most similar to an incremental neural network parser that explicitly represents the syntax of a statement with architectural arrangements. It is interesting that in the above strategies, the notion of semantics is tightly tied to syntactic inference. It is seen for instance that determining the role of a token relies on mapping it to a previous occurrence of the same token so that the kind of knowledge it conveys in a statement is required in order to place it into the parse tree.

Delegation of parsing subtasks is also seen in above architectures, although not in the same way as is accomplished by the NGN. The SPEC in particular breaks down parsing in a dramatically different way than does the NGN by further decomposing the process of mechanical parsing to functional equivalents provided with the three smaller enclosed neural network architectures (the feed forward network, the recurrent network and the RAAM).

2.11 Conclusion for Neural Network Parsers

Neural network parsers may be tasked with varying degrees of syntactic structuring, case role labeling and semantic inference. These tasks are accomplished either incrementally by processing a statement one token at a time or holistically by processing an entire statement taken together. The processing of molecules for classification and regression tasks requires the combination of both syntax structuring and semantic learning and generalization. Neural network parsers that are more capable of syntactic structuring such as the HCP must be

modified to process latent semantic data before they can accommodate molecular discrimination tasks. Devices that are designed with the sole intention of case role labeling such as the SPEC and Hebbian parser are not foreseeably applicable to molecular recognition as the way data is processed and output is generated is already convoluted and well structured. Neural network parsers that are capable only of simulating semantic inference such as the simple recurrent network lack the content specificity needed for long strings belonging to more complicated formal languages.

In the next section, we visit two graph based QSAR approaches. One is based on structuring a neural network based on a molecules' traversal and the other utilizes a kernel method that calculates the difference between two molecules based on their graph topologies.

2.12 Graph Based Machine Learning Approaches in QSAR

Presented in this section are other machine learning approaches similar to the NGN that have also been used in QSAR problems. These approaches are different than the ones discussed earlier in that they, like the NGN focus on a structural graph-based description of a molecule. Two approaches are discussed in detail here, one which compares the similarities of molecules given a graphical molecular kernel function enclosed in a potential-support vector machine (P-SVM) and another which maps a homogenous recursive artificial neural network onto a graph representation of a molecule.

2.12.1 Potential-Support Vector Machine with Molecule Kernels

A kernel method (Mohr et al., 2008) is described first. Support vector machines are similar to artificial neural networks in that they handle vectors as inputs and outputs and are capable of universal function approximation. SVMs are different in that they host kernel functions which allow for the classification of vectors. Abstractly, this is done by converting inputs into n-dimensional vectors. A hyperplane is then mathematically induced which

separates these input vectors. This separation is the decision threshold which determines the classification of a given vector. Like a Neural Network equiped with one or more Hidden Layers, the SVM is capable of solving problems which describe some linear inseparability.

A Potential-Support Vector Machine is a SVM which uses a kernel function that can differentiate between two data points regardless of the kind of data as long as the output is some consistently calculated difference. The novel feature of the discussed approach is the Molecule Kernel.

In this approach, a molecule is defined as a graph, so that atoms are mapped to vertices, and bonds are mapped to edges. Atoms (vertices) are tagged with additional spatial coordinates. Bonds (edges) are labeled with an integer [1, 4] representing single, double, triple and aromatic bond, respectively. Both the graph structure and spatial-topological information are expressed in the above scheme. Molecules are compared given an optimal alignment algorithm. The notion of a bipod is introduced as three atoms in a molecule and the two bonds connecting them. Abstractly, the algorithm superimposes two molecules together so that a single bipod from each sees the translation and rotation of one entire molecule onto another. This is done for all bipods for the two molecules. The similarity score for the molecule is given the maximum value in the resulting set of bipod alignments. It is this similarity score that is used as the kernel function for the P-SVM in this approach.

It is found that the runtime complexity of this method is $O(n^4)$, so that n is the number of atoms considered in the problem; this is based on the bipod alignment definition above.

The NGN-QSAR approach is different from the above molecule kernel P-SVM approach as the NGN does not capture absolute spatial coordinates, but rather substitutes this information with the relative position of chiral and isomeric bond positions. As well, the graph that the NGN constructs features input vertices that are mapped to any kind of molecular information including atoms, bonds and abstract entities such as counts, charges and rotation. Edges are mapped to vector processing weight layers. While the kernel approach does scale to the fourth degree, it is always guaranteed to provide a set of outputs. The NGN has no such guarantee as it may not converge, but could also offer convergence within only

a few epochs since it does not define an exhaustive comparison between each possible pair of molecules in a given problem. In terms of space efficiency, this is a real bonus considering that the corresponding disk bound weight layers of a neural network are of a constant size based on the grammar, and not the number of molecules.

The price of the performance gain is that the molecule kernel P-SVM is currently capable of solving a broader set of problems than the NGN-QSAR. Particularly, larger data sets in regression problems completed with the P-SVM have been particularly difficult for the NGN to solve.

Other kernels used in P-SVMs for molecule graphs include Decomposition Kernels (Ceroni et al., 2007) which calculate difference based on groups of bonded atoms in each molecule, and Convolution Kernels (Ralaivola et al., 2005) which calculate difference based on a count of paths that traverse particular sequences of atoms via bonds.

2.12.2 Recursive Neural Network Mapped onto Molecular Graphs

One further approach tried by (Walsh et al., 2009) is discussed here. In this approach, the node layers of a recursive neural network are mapped onto the atoms of a molecule, and the weight layers are mapped onto the bonds. The network features the use of a homogenous weight layer so that each weight layer that is dynamically assembled onto the molecule graph is a reference to the same 2D matrix. This weight layer is broken up so that some of the units accept an input vector representing an atom, and the other units accept recurrent output from antecedent layers. Molecules are represented a number of times equal to the number of atoms in that molecule so that each atom has a turn as root. There is no exponential explosion of valid traversal trees in memory because each root is connected to only the first valid traversal. A single representation of a molecule is then a supertree whose root is connected to all traversal trees for a given molecule. Cycles are dealt with as follows. A given traversal may stop at a terminal vertex of a graph (the final atom in a branch of atoms) or when an atom has already been accounted for (an atom is seen the second time within a loop).

As in other dynamic neural network frameworks, function approximation and training algorithms can be applied once assembly is complete.

The network architecture is different from the NGN in that several (though not all) permutations of atoms are represented in the assembled dynamic network whereas a single canonical version of a molecule is used in the QSAR-NGN approach. This offers Walsh's approach the ability to discover on its own, the best representation or combination of representations.

2.13 Chapter Conclusion

In this chapter, we have discussed four major background topics of interest. We have revealed the cultural and technological basis of the NGN from the vantage point of Neural Network architectures and from that of formal language parsers. We have also visited the Neural Network Parsers developed by other researchers and have finally described other graph based approaches in QSAR.

In the next chapter, we formalize the NGN mathematically and address its implementation along with the parsing of statements from an example arithmetic formal language.

Chapter 3

Neural Grammar Network Implementation

A discussion of all prerequisite concepts have been undertaken and the reader is prepared for NGN implementation details.

3.1 Chapter Introduction

An extensive description of how the NGN is implemented is found in this chapter. This is interleaved with a mathematical formalization. The mathematical formalization included is designed to be comprehensive with respect to the mappings between conceptual components of the NGN.

3.2 Overview

We overview this extensive chapter as follows. We discuss a simple and familiar arithmetic expression legal in the C programming language that will be used as an example throughout this chapter. This is followed by a revisitation to mechanical formal language parsers as

a topic of implementation with respect to the NGN. Weight matrices and bias vectors are discussed. The NGN is discussed with respect to its runtime lifecycle followed by the three formalized neural network algorithms as it pertains to the NGN; feed forward, back propagation (gradient finding) and weight update. Finally, the automated NGN source code generator script is discussed.

3.3 An Arithmetic Example

An arithmetic example statement and grammar is shown below and its treatment by the NGN will be used as an example for the general operation of the NGN throughout this chapter. The grammar used is derived and simplified from the C programming language grammar (Kernighan et al., 1988).

Example 3.1. An arithmetic C statement.

```
9 + 6 * 4 / 2 - 5 ;
```

This is a statement of familiar syntax which demonstrates the use of operator precedence. In a valid parse of this string, one would expect that the multiplication and division tokens are processed first along with their operands before the addition and subtraction tokens. One also expects that operands shared by the operators of the same strength should be first processed by the left most legal substatement. Finally, the statement delimiting semicolon should be processed last as it does not contribute any intermediate value to the semantics of the underlying math. All of the above structuring features can be defined by the syntax of the grammar.

An example simplified grammar accompanying the C statement discussed follows (Kernighan et al., 1988).

Example 3.2. A simplified C grammar.

```
Expr → OpAdd SymSColon // ‘‘Expr’’ is the start symbol
```

```
SymSColon → ‘;’ // ‘;’ is a terminal symbol

OpAdd → OpAdd SymPlus OpMult | OpMult

// sequence of internal symbols in right side

SymPlus → ‘+’ | ’-’

OpMult → OpMult SymTimes Digit | Digit

SymTimes → ‘*’ | ’/’

Digit → ‘0’ ... ‘9’
```

How the NGN treats the above statement is largely dependent on its internal parsing components. We discuss the behaviour of these components next.

3.4 Parsing Revisited

Here, we discuss parsing and discuss the mechanical parsing technologies chosen for the NGN. An expansion to earlier explanation is given and treatment of specific limitations of the technology is discussed.

3.4.1 Flex, GNU Bison and Formal Grammars

The NGN software can be thought of as a GNU bison parser (GNU bison, 2008) with a flex tokenizer (flex, 2008) that is tightly coupled with an in-house developed general neural network library.

Flex and GNU bison are used to tokenize and build the parse tree respectively. Tokenization is the recognition of deterministic string fragments that correspond to legal formal language statement particles. Based on these recognized tokens and a formal grammar, a neural network tree is constructed so that activation layers (node layers) correspond to

the symbols of the grammar and recycled weight layers are used to connect them. After the tree is built, the tokens are vectorized and are fed into the network inputs as "1.0". Weight layers may optionally expand to more than one statement. In this case, symbols corresponding to the selected expansion statement connect with respective node layers for each symbol; symbols corresponding to the unselected alternative statements are fed in as "0.0".

At this point, the fully assembled NGN is available for training or recognition tasks.

After these tasks are complete, the NGN may be disassembled so that node layers and their contents are destroyed, but weight layers are saved as they represent the cumulative sessions of training experienced by the NGN.

Formal grammars define the legal syntax of a language. Grammars must be in a form that can be parsed by a left-to-right one-look-ahead (LALR(1)) parser in this implementation. This form is compatible with GNU bison generated parsers. Grammars in this form consist of a set of symbols and productions that relate a single symbol on the left side to a statement on the right side. A statement on the right side is a sequence of symbols. This form is otherwise known as Backus-Naur Form as discussed previously. Iteratively reducing statements matching right hand expressions into symbols on the left hand side is coarsely the activity of a the parser. Symbols may be terminal tokens representing units of input, or internal symbols which correspondingly expand to other statements.

It should be noted that grammars in Backus-Naur Form may contain legal statements that require more than one look ahead token to process. This is not a problem if this statement cannot be confused with any other statement. When two or more statements contain the same long sequence of symbols so that only one token differs in the middle, then an ambiguity arises. To repair this ambiguity, the statements involved are translated into Greibach Normal Form (GNF) wherein such similar statements are reexpressed as a telescopically expanding series so that each step in the series only contains one terminal token. Note that more powerful though less efficient parsers which handle more than one

look ahead token may accommodate this problem without resorting to editing the grammar. Such parsers were not tried with the NGN.

A GNF Grammar contains statements which appear like the following production rule.

$$\text{InternalSymbol}_i \rightarrow \text{Token} \cdot \text{InternalSymbol}_j$$

This is true such that 'InternalSymbol' may be any internal symbol. The resolution of this grammar is shown below, so that the two statements involved with conflict are rewritten in GNF form. Note that the remainder of the grammar may stay the way it is unless the resolution leads to new conflicts. The '|' operator indicates alternative legal parses. This also demonstrates the appearance of a telescopic expansion.

A BNF grammar is shown below with a statement irresolvable by a LALR(1) parser due to the single look ahead token limitation. The token 'C' in the former possible production is replaced by the token 'Z', requiring resolution in the latter production.

$$S \rightarrow \text{`}A\text{''}B\text{''}C\text{''}D\text{''}E\text{'}|\text{`}A\text{''}B\text{''}\mathbf{Z}\text{''}D\text{''}E\text{'}$$

The GNF equivalent which allows LALR(1) parsers to operate.

$$S \rightarrow \text{`}A\text{'}B_{\text{more}}$$

$$B_{\text{more}} \rightarrow \text{`}B\text{'}C_{\text{more}}|\text{`}B\text{'}Z_{\text{more}}$$

$$C_{\text{more}} \rightarrow \text{`}C\text{'}D_{\text{more}}$$

$$Z_{\text{more}} \rightarrow \text{`}Z\text{'}D_{\text{more}}$$

$$D_{\text{more}} \rightarrow \text{`}D\text{''}E\text{'}$$

3.4.2 Arithmetic Example Parse Tree

Shown in Figure 3.1 is the parse tree for the C arithmetic statement (Example 3.1) given the simplified formal grammar (Example 3.2), where tokens are indicated in single quotes, and internal symbols are shown in bare text. The single start symbol of the grammar is 'Expr'; this is the single root of the parse tree. Notice the multiple use of symbols such as 'OpAdd' and 'OpMult' as well as the use of their alternative parses. The parse tree is a constructed representation of the input statement that well describes hierarchy and order of operations by the differing heights of the symbols on the tree. Some symbols in the parse tree are referenced more than once. This recursive dynamic structure is an important feature used by the NGN.

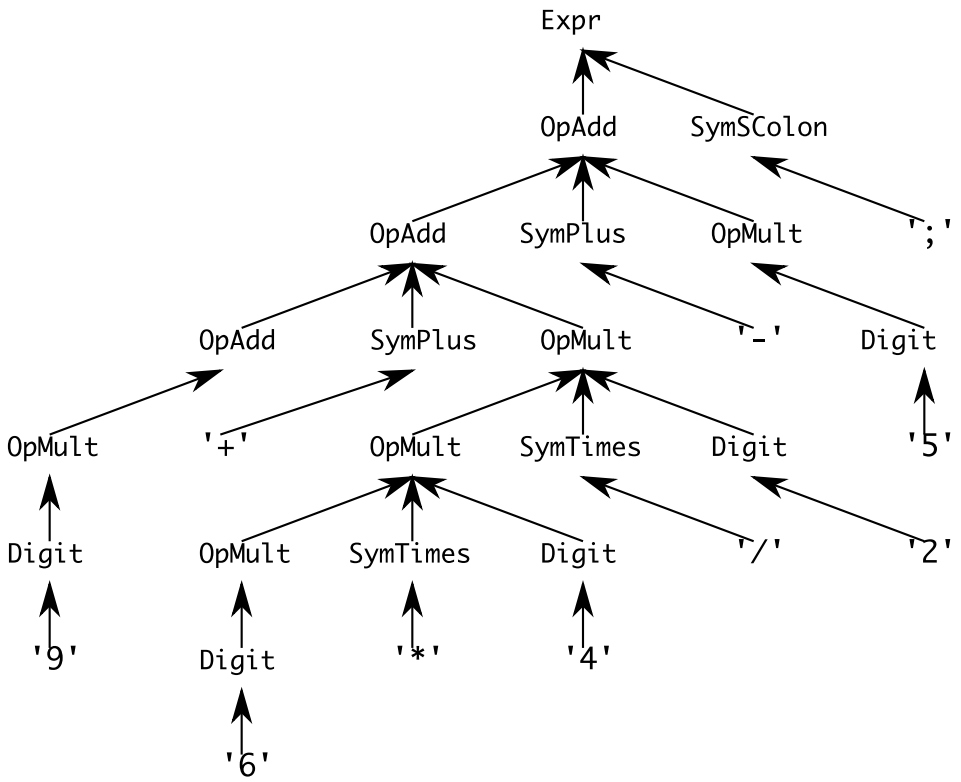


Figure 3.1: Example 3.1 structured against Example 3.2 as a parse tree by a LALR(1) parser.

3.4.3 Formal Language Math for the NGN

Along with the input string, the formal grammar provides the consistent and deterministic structure required for the operation of the NGN. Below is formalized how a grammar is defined.

$$G =< V, \Sigma, s, P >$$

The grammar (G) is a time invariant tuple of internal symbols (V), terminal symbols (Σ), a start symbol (s) and a set of production rules (P). The constraint $s \in V$ applies.

A production rule maps a right-hand statement (a sequence of symbols) onto a left-hand reduction (a single symbol). Each production rule is expressed in the form $\{A \rightarrow \{\alpha^+\}\}$ so that $+$ indicates the concatenation of one or more symbols, that $A \in V$ (reductions may only be internal symbols) and that $\alpha \in (V^+) \cup (E)$ (elements of a statement may only be a sequence of internal symbols or a single terminal symbol).

Parsing a string according to the grammar results in a parse tree. Because both direction and the notion of a root are implied, the tree used is formally known as an arborescence. The arborescence defined features additional constraints which ease mapping productions of grammar onto sets of children in the arborescence. With these additional constraints, the term Ordered Child Arboresence (OrCA) is used. The root in this structure is defined as a unique and universal directional source.

Whereas an arborescence is defined as a tuple of nodes (N) and edges (E), an OrCA (T) is defined as a tuple of nodes (N) and ordered child edge sets ($\mathbb{E}$). Each ordered child edge set is an ordered pair whose first element is an OrCA node and whose second element is an order-sensitive sequence of other OrCA nodes. The first element is the parent (upstream), and the second element is the sequence of children (downstream). The OrCA is time sensitive as it relies on the time sensitive input string to derive its structure as well

as the fixed grammar.

$$T(t) =< N(t), \mathbb{E}(t) >$$

The ordered child edge can be thought of as follows where the symbol $\in_{substr}$ indicates that a token can be found in a string (string membership).

$$\mathbb{E}(t) = \{(u \rightarrow v) | u \in_{substr} N(t) \wedge v \in (N(t))^{+}\}$$

The definition of the OrCA is completed below by describing how to derive a set of edges E from a set of ordered child edge sets $\mathbb{E}$, how to derive a path $\mathbb{P}$, the absence of cycles $(u, v) \in \mathbb{P} \Rightarrow (v, u) \notin \mathbb{P}$, and the notion of a root r. This is done to provide the stipulations needed to complete the mappings of the NGN topology on the OrCA structures.

Edges are described as follows.

$$E(t) = \{(u, v_i) | (u \rightarrow v) \wedge v_i \in_{substr} v\}$$

A single edge maps onto a connection between a grammar internal symbol and one of the symbols in one of the statements it expands to. Edges build to paths as follows.

$$\mathbb{P}(t) = \{(u, v) | (u, v) \in E(t) \vee [(u, w) \in E(t) \wedge (w, v) \in \mathbb{P}(t)]\}$$

The notion of a path is needed to build the next stipulations, first that no cycles may exist in the OrCA.

$$\forall (u, v)\ (u, v) \in \mathbb{P}(t) \Rightarrow (v, u) \notin \mathbb{P}(t)$$

Second, paths are also used to assert that there exists a unique and universal root.

$$\exists r \forall v \ (r, v) \in \mathbb{P}(t) \Rightarrow r$$

The acyclic and single root properties importantly formalize the desired bounds on the mapped neural network layer topology of the NGN. Cycles should not be confused with recursion; the NGN shares processing weights for recursion, rather than looping activations recurrently (where cycles would be needed). Notice that the root of the OrCA eventually resolves to the start symbol of the grammar as well as the output layer of the assembled NGN.

An input string $s(t)$ is mapped to the terminal symbols of the grammar as follows. An input string is a concatenation of input tokens ($s(t)_i$). Note that the string is time sensitive in that the input string is a different item to parse at each time step.

$$s(t) = s(t)_0 s(t)_1 s(t)_2 ... s(t)_{|s(t)|-1}$$

The above is true such that $s(t)_i \in \Sigma$. The following notation is used to indicate concatenation hereafter. It is read "*The concatenation of tokens at time t from zero to the length of the string*".

$$s(t) = \mathop{\Big\Vert}_{i=0}^{|s(t)|} s(t)_i$$

In Example 3.1, each of the tokens is found in the right hand of some production rule in the above grammar.

3.5 Weight Initialization

In this section, we discuss the initialization of the NGN as occurs for each exemplar. This process is done only once before parsing any statements. Assembly is however repeated each time a new exemplar is read to the NGN.

Like many neural network topologies, the NGN consists of activation layers and weight layers. Activation layers are created in memory on demand while weight layers are saved and re-used for each exemplar. This allows the collection of weights to reflect the total training experienced by the NGN. Activation layers correspond to grammar symbols, while weight layers correspond to the production arrow that links a grammar symbol to all of the symbols in each statement it can possibly expand to. A statement that is an alternative expansion for a symbol that is not used is attached to a vector flushed with '0.0'. This is explained in detail when we introduce Figure 3.5. Tokens are converted to activation vectors of length one, with the value '1.0'.

Along with the grammar productions, each symbol must be assigned a specific number of processing units by the user. For Example 3.1, this is done here.

Example 3.3. Processing unit assignment for example C grammar.

```
Expr ← 1

OpAdd ← 3

OpMult ← 3
```

The output layer `Expr` gets only one output unit in our example. Note that the number of output units depends on the problem being solved and the encoding used to describe the solution. Since there is only one output node in our example, example problems that it can encode is whether the arithmetic answer is odd or even, whether the arithmetic answer is high or low and other items that can be described in by a binary or boolean status.

The hidden layers `OpAdd` and `OpMult` get three units each. This is generally determined experimentally based on performance, favouring a practical neural network convergence rate as well as retrodictive accuracy (on the training set), predictive and generalization capability (on the test set). Notice that some data compression occurs as the activation values transit between layers in that the number of units in each kind of layer is fixed.

Notice that the symbols that map hidden layers directly to input units are not assigned a number of units. These symbols `SymSColon`, `SymPlus`, `SymTimes` and `Digit` are automatically assigned a number of units equal to the number of possible inputs they can accommodate. In our example, this entails the following mapping.

Example 3.4. Processing unit autoassignment for Hidden Layers connecting Input Vectors.

```
SymSColon ← 1

SymPlus ← 2

SymTimes ← 2

Digit ← 10
```

When an input token is vectorized, it is fed to a different unit within the vector of the hidden layer it feeds into and is given the activation '1.0'. It is this position that allows the NGN the ability to distinguish between different alternative symbols allowed. All other positions that are not connected are flushed with '0.0'. This is clarified in the diagram Figure 3.3.

3.5.1 Implementation

A general neural network library was implemented in the C programming language. The architecture was implemented so that the activation vectors and weights are conceptually different data structures called node layers and weight layers respectively. Node layers

are used to form the tree structure of the network so that a parent node layer may have many children node layers. Corresponding weight layers are slotted in which match the dimensions and identity of a given parent to its child. In this way, weight layers benefit the system by representing accumulated training while node layers are free to take on new values corresponding to a specific parse or novel statement of input.

Adaptations are implemented on the feed forward and training algorithms as follows. As a tree, care must be taken to ensure that the overall result of processing data is equivalent to a neural network whose layers are arranged in a simple stack. In feed forward, a post-order or depth-first traversal is done on the tree so that the deepest layers are visited first and their activations calculated. The math and application of the transfer function is identical. Note that several input layers may occur at different heights of the network, each attaching at a leaf of the tree. The gradient descent algorithm in training consists of the expected two phases, the back propagation of error and the forward propagation of weight updates. The error propagation is conducted as a pre-order or breadth-first traversal and the update phase is conducted as a depth-first traversal as in the feed forward pass. The expected math functions are applicable for the gradient descent algorithm here. It should be noted that node layers also contain the needed data structures to keep track of the gradient descent algorithm and are freely instantiated and destroyed on demand along with the rest of the node layer.

Different input statements create different trees. Weight layers are assembled dynamically based on these trees. A single tree may have multiple references to the same weight layer allowing for recursive processing of input. As well, a single tree may contain many references to different weight layers allowing for heterogeneity.

With preliminary tests, the sizes (magnitudes) of initial random weights were examined. Conventionally, small random values are used to break the bias of the weights so that each unit converges on different final destination values. It was discovered that larger random weights tended to cause the network to converge more quickly. The use of large random weights shortcuts the work needed by the gradient descent algorithm by essentially

introducing a random search element to training; combinations of weights that do converge, do so more quickly and non-converging combinations can be detected and rejected earlier, with fewer epochs. Large initial random weights are defined by introducing a blindspot around zero within which no initial values can fall. The randomly obtained value is simply added to the absolute value of the blindspot and thus pushed away from zero. This random searching was inspired by the simple recursive random search net employed by Hochreiter and Schmidhuber (1996).

3.5.2 Hidden Layer Units in the NGN

Figure 3.2 shows the reservoir of weights as allocated in memory prior to any parsing by the NGN. The node layers are dashed in these diagrams to indicate that no node layers are connected to the weight layers shown. The dotted node layers indicate the number of units that are connected to the weight matrices. Notice that a certain child symbol may occur more than once in the node layer beneath any weight layer. This can happen if a parent symbol expands to a statement that syntactically requires more than one occurence of a child symbol, or if a parent symbol expands to one or more statements that have an occurence of the same child symbol. The latter occurs in the grammar we demonstrate as the symbol `OpAdd` expands to `OpAdd SymPlus OpMult` or `OpMult` and as the symbol `OpMult` expands to `OpMult SymTimes Digit` or `Digit`. In the above expansions, the logical 'or' separates alternative statements that contain a repeated child symbol. The allocation of the reservoir of weights does not change between parses as only the values of those weights change. A parsed statement can result in an NGN that makes zero to many references to each weight layer distributed throughout the entire tree. Training entails that changes are made to the values of the weights in the reservoir. Figure 3.3 extends the reservoir diagram by expanding each hidden layer specialized at receiving activation from input vectors.

This reservoir is populated by the same weight layers throughout every parse of the NGN. Multiple reference in the same parse and between parses are used to accumulate data. In Figure 3.2, the dashed activation layers indicate the number of units per produc-

tion symbol and the production symbols in every statement it expands to. Each possible alternative statement is indicated. The NGN is able to differentiate between different possible expansions by the positions of the incoming activation vector as it is connected to the weight layer.

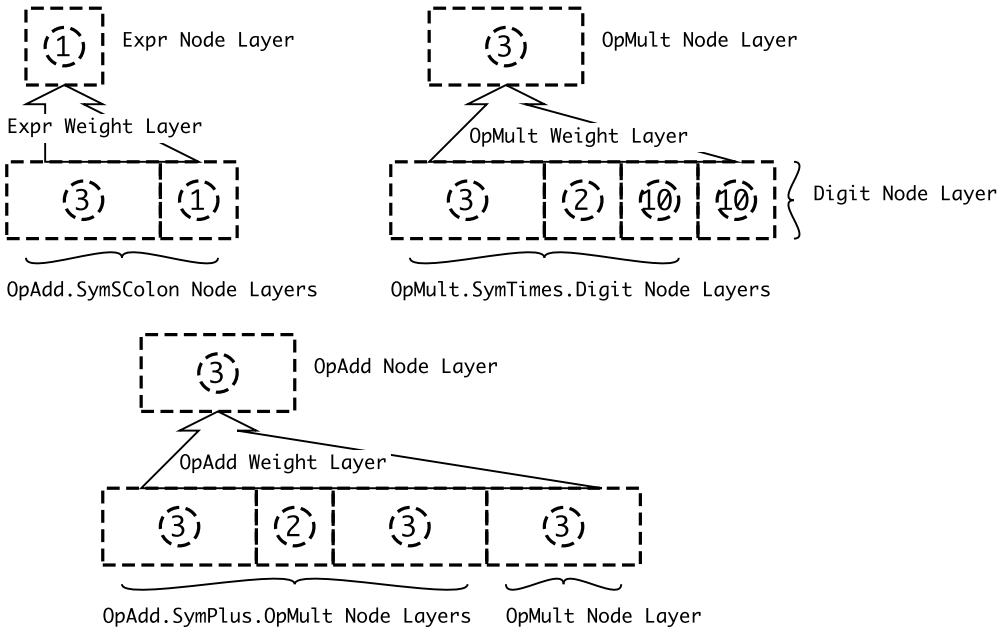


Figure 3.2: A weight layer reservoir. This is an abstract representation of the weight layers in memory.

In Figure 3.3, the number of processing units are shown for each hidden layer that specializes at receiving input symbols only. Each input symbol is vectorized as a length one vector of value '1.0'. Like the previous figure emphasized, it is the position of the incoming vectorized token which is used by the NGN to differentiate amongst the tokens. All other potions are flushed with '0.0'.

3.5.3 Mapping Weights onto Formal Grammar

How weight layers relate to the grammar is discussed here. The neural network weights are re-used between different input strings for the same formal grammar in the same session of training. In order to accomplish this, we conceptually map a sequence of weights onto the

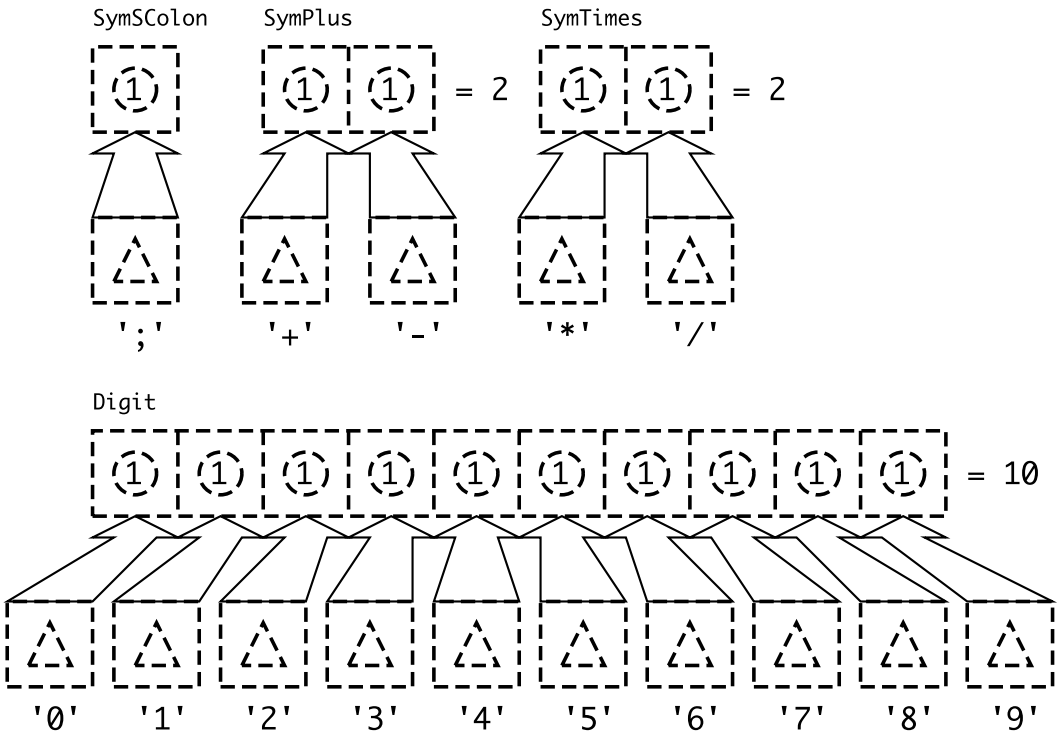


Figure 3.3: This is a continuation of Figure 3.2, expanding on the hidden layers connecting to input layers.

grammar's production rules. The term weights (W) more precisely consist of weights (w) and biases (b). Weights differ between time steps, but each temporally successive matrix of weights is related to its antecedent by the gradient descent training algorithm used. We use a back-propagation training algorithm to change the value of weights from one time step to the next.

$$W_p(t) = \{< w_p(t), \vec{b}_p(t) > | p \in P\}$$

The number of hidden units corresponding to internal symbols is arbitrarily defined by the user of the system. We describe this with the following function (l).

$$l : \alpha_i \rightarrow \mathbb{N} > 0$$

The number of hidden units in the parent layer is taken as the number of rows of a weight matrix while the sum of the number of hidden units of the child layer is taken as the columns of a weight matrix.

$$w_p(t) \in \mathbb{R}^{(l(A))\cdot\left(\sum_{i=1}^{k} l(\alpha_i)\right)}$$

The number of bias units is given by a column matrix whose height is equal to the number of hidden units corresponding to the parent.

$$b_p(t) \in \mathbb{R}^{l(A)}$$

Figure 3.4 demonstrates such a mapping taken from Example 3.1 for clarification. Notice that the number of rows of the weight matrix equals the number of hidden units corresponding to the parent symbol while the number of columns corresponds to the total number of hidden units of the children. The column matrix has as many rows as the parent has hidden units.

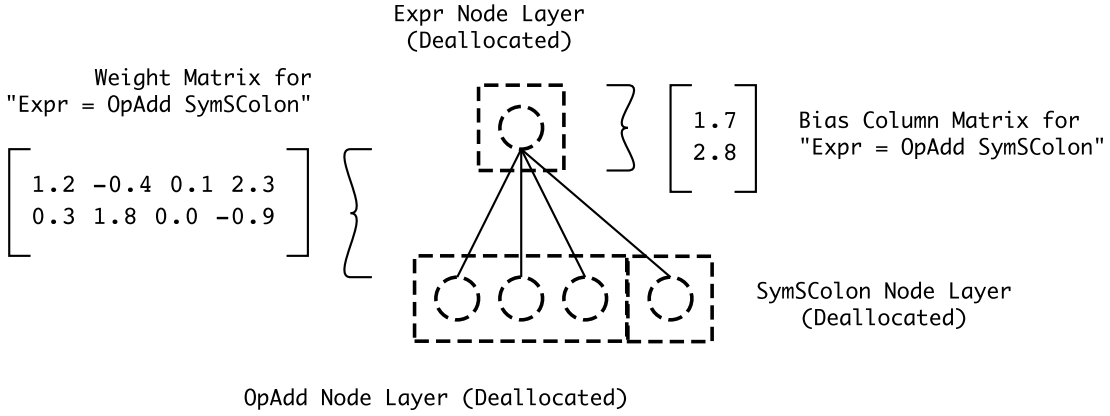


Figure 3.4: A diagrammatic overview describing the mappings of weights onto a grammar in the NGN using the internal statement "Expr → OpAdd SymSColon".

3.6 Network Assembler

In this section, we discuss the assembly of the NGN during parsing and address its relationship with the input string.

3.6.1 Parsing and Dynamic Assembly

An instance of an NGN is assembled based on the corresponding parse tree of its input statement and grammar. Weight layers correspond to the connections in the parse tree between a symbol and the statements it expands to. Each weight layer is referenced one or more times from the same reservoir of weights that was initialized prior to any parsing. Activation layers are destroyed after each parse but changes made to weight layers in training are saved in the reservoir of weights.

3.6.2 Arithmetic Example Parsed as an NGN

Figure 3.5 shows the assembled NGN for Example 3.1 with the grammar Example 3.2. Notice that the order of operations is preserved and expressed simply by the syntactical structure of the grammar; multiplication and division occurs lower down in the tree than does addition and subtraction. Weight layers are labeled by the parent symbol and the child statement it expands to. Other possible statements are indicated by dashed figures. A weight layer receives a "0.0" for each unit from such dashed alternative unselected expansions, and receives an activation $[0.0, 1.0]$ for allocated activation layers. Input vectors contribute an activation of "1.0".

Input vectors for each token are shown in Figure 3.6. These vectors are created by the flex component of the NGN software when it encounters an input token. These vectors are structured as input activation layers into the NGN. The final NGN structure depends on the formal grammar as well as the input tokens. The sequence of input tokens is as important as which tokens are selected. Different legal sequences of the same tokens will

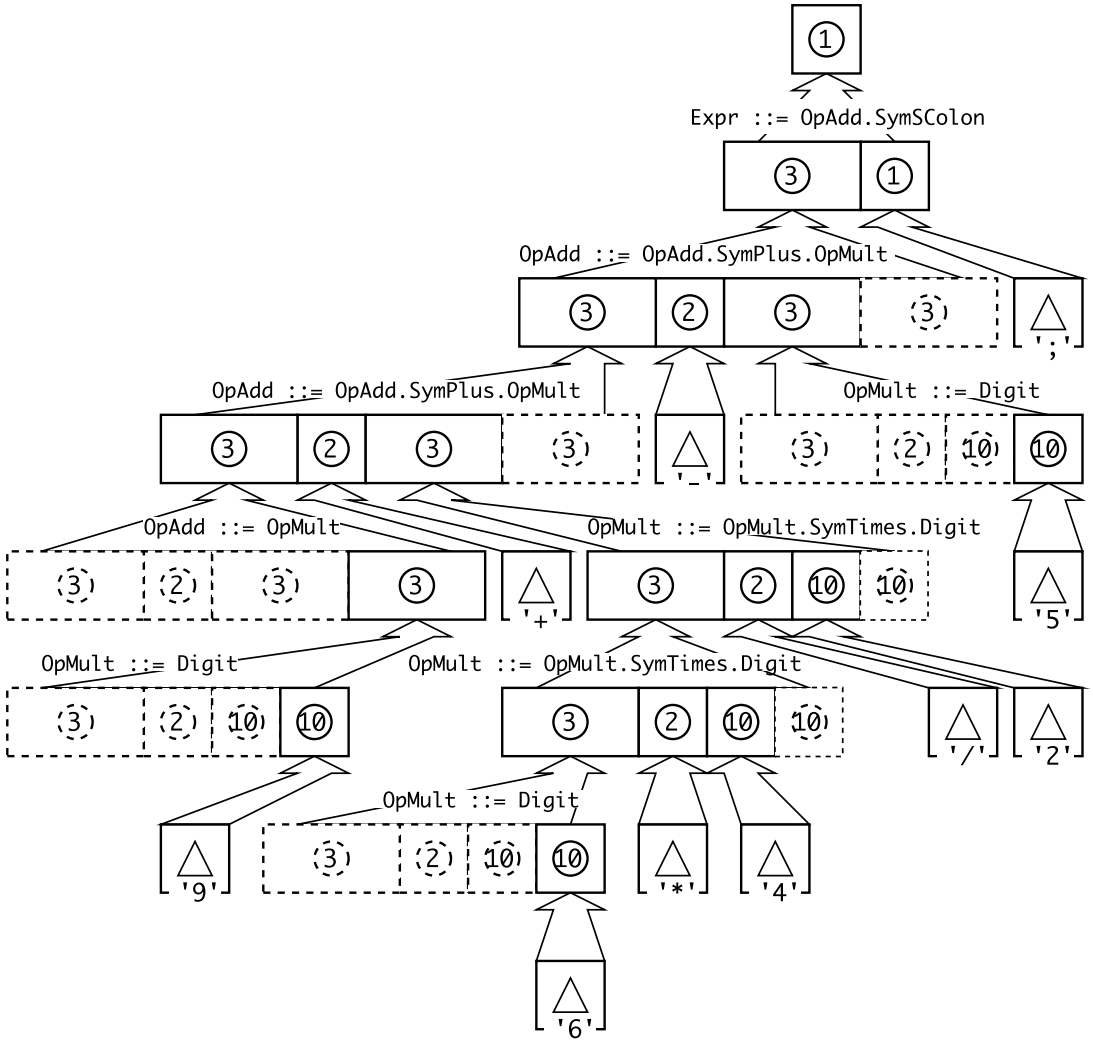


Figure 3.5: The assembled NGN for the statement Example 3.1 on the grammar Example 3.2.

lead to different NGNs being generated derived from the different corresponding parse trees. The NGN will only operate on statements conforming to the syntax of the grammar.

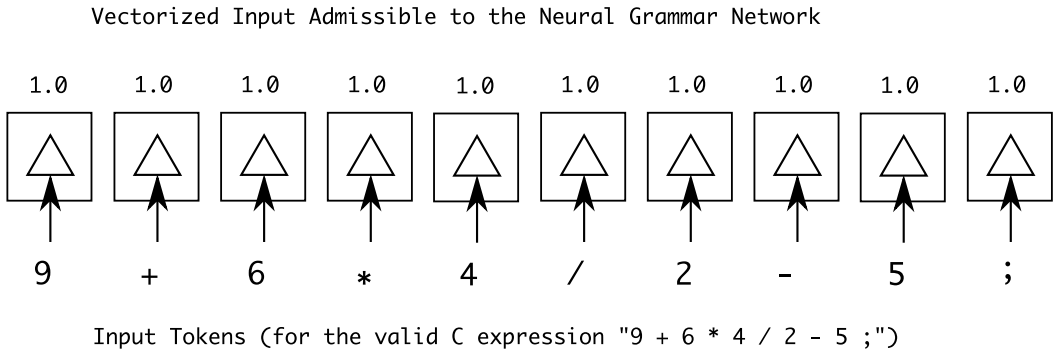


Figure 3.6: The flex lexicographical analyzer converts tokens into input layer activations.

3.6.3 Formal Mapping Activations for the NGN

The input vector for any input layer of the NGN is always a real value 1.0. The difference between formal strings is described by the structure of the NGN rather than its inputs alone. The reliance on position to determine symbol assignment is a generalization of the "one-hot" encoding scheme. In our implementation, the **flex** generated lexicographic analyzer performs the input vector mapping and returns a reference to the correct layer. The lexicographical conversion for Example 3.1 is illustrated in Figure 3.6.

$$\vec{a}_s = (1)$$

$$\textbf{flex} : \Sigma \rightarrow \vec{a}_s$$

Neural Network activation vectors are the final piece to integrate into the NGN puzzle. Below, σ translates an OrCA node into a grammar symbol and l translates the returned grammar symbol into the number of hidden units found in that node layer. The activation

vector ($\vec{a}$) for a given node layer is thus expressed as follows.

$$\vec{a}(t) \in \mathbb{R}^{l(\sigma(N(t)))}$$

The activations of the node layer assigned to the root symbol is the final output vector of the NGN.

3.7 Feed Forward

To calculate the activations to a parent from its children, two items are considered. They are first, the math function needed to relate how weights and activations are related; and second the computational algorithm performed on the gross NGN structure.

The math function which combines the activation vectors of children and the weight layer between the children and the parent is defined as the function **activity** below. The output vector is given by a sigmoid transformation of each element of the row matrix resulting from a cross product of the weight layer and input activations.

The variable 'q' is the number of productions that expands from the symbol 'u'.

The function **activity** is defined as follows.

$$\vec{a}_u(t) = \mathbf{L}\left(\left[\overset{q}{\underset{i=1}{C}}(w_{\sigma(u\to v_i)} \times \vec{a}_{v_i}(t))\right] + \vec{b}_{\sigma(u)}\right)$$

The logistic function **L** is the following.

$$\mathbf{L}(x^{(i)}) = \frac{1}{(1+e^{-x^{(i)}})}$$

The feed forward algorithm can be described as an application of the above math in a depth first or post-order traversal of the NGN so that values related to the input layers are calculated first.

A back propagation learning (BPL) algorithm involves the percolation of error up the tree in a breadth first or pre-order traversal of the NGN, while the weight updating feed forward pass is again a post-order traversal. The BPL is covered in detail next.

3.8 Back Propagation

To increase visual clarity, the remaining formalization implies the use of the previous σ function where appropriate; for instance where previously $w_{\sigma(u\to v)}$ would indicate the weight layers connecting the parent node layer u to the child node layers v, below this is simply abbreviated to $w_{(u\to v)}$. Note also that where an iteration over a collection of values is implied, the notation that indicates the vector or matrix is changed as follows. The indication of the weight layer from parent u to child v as $w_{(u\to v)}$ is changed to an iteration by writing $w^{(ij)}_{(u\to v)}$ meaning to iterate over the parent nodes i in an internal minor loop then to iterate over the child nodes j in an external major loop. A similar notation change is used for vectors; for example, to iterate over each value of the vector $\vec{a}_v$, one writes $\vec{a}^{(j)}_v$.

The BPL algorithm operates by pushing parametres in a direction that minimizes error in output. Abstractly, the difference between the intended output of the system is compared with the experimental system output with the intention to find parametres that are responsible for defects in the prediction, and by what gradient (i.e. whether the parameter is too positive, too negative or incorrectly oriented in higher dimensional spaces). Each parameter is adjusted in a direction which reduces error.

The BPL algorithm is performed in two steps, first the gradient of error is found, then weight changes are calculated and added to the weights. As in the feed forward above, gradient finding is presented first with the math required at each level of height in the NGN OrCA, followed by an algorithm which visits each node layer in sequence.

There are two different equations used to find the error gradient $\vec{\delta}$ depending on the location of the node layer being operated on. The error for the output layer is straight forward, defined with respect to the system output $\vec{a}_v$ and the target $\vec{a}_T$ activation pattern.

The error for each hidden layer $\vec{\delta}_v$ requires assigning the blame for error based on its own activation $\vec{a}_v$ and *its nodes' contributions to its parent's error*. This blame is expressed in terms of the parent's own gradient $\vec{\delta}_u$ and the weights connecting this node with its parent $w_{(u \to v)}$.

Notice that the bias terms need not be addressed in the gradient finding step.

The function **gradient** is defined as follows for an output layer.

$$\vec{\delta}_v^{(j)} = \left[\left(\vec{a}_v^{(j)} \right) \left(1 - \vec{a}_v^{(j)} \right) \left(\vec{a}_T^{(j)} - \vec{a}_v^{(j)} \right) \right]$$

The function **gradient** is defined as follows for each hidden layer.

$$\vec{\delta}_v^{(j)} = \left[\left(\vec{a}_v^{(j)} \right) \left(1 - \vec{a}_v^{(j)} \right) \left(\sum_{i=1}^{|u|} \vec{\delta}_u^{(i)} w_{(u \to v)}^{(ij)} \right) \right]$$

The above two equations are usually applied as a breadth first traversal so that the the gradients for the output layer are calculated first, and the gradients for the hidden layers connecting the input layers are calculated last. Finally, a weight update pass is done.

3.9 Weight Update

The **delta** function generates the following mathematical transformation for weights and biases where η is the learning rate and α is the momentum coefficient. Going from left to right, we address each variable found in the below equations. The variable Δw represents the calculated change needed for a given 2D weight layer matrix at a specific instance in the NGN. This item is subscripted $(u \to v)$ to specify which 'parent $\to$ child' node layers it is connecting while the superscript (ij) indicates an iteration of the units in the parent

and child layers respectively. The superscripts (i) and (j) are consistently used to indicate iteration as above. The variable $\Delta\vec{b}$ is the calculated change for a given bias column vector belonging to a given weight layer instance in the NGN. The time variable (t) indicates the present training step, so that the present result of the function depends on the previous result. The gradient vector of the parent layer $(\vec{\delta_u})$ calculated in the gradient finding step above is used in the present calculation along with the activation vector of this layer $(\vec{a_v})$ calculated in the feed forward step.

$$\Delta w^{(ij)}_{(u\to v)}(t) = \eta\vec{\delta}^{(i)}_u\vec{a}^{(j)}_v(t) + \alpha\Delta w^{(ij)}_{(u\to v)}(t-1)$$

$$\Delta\vec{b}^{(i)}_u(t) = \eta\vec{\delta}^{(i)}_u(t) + \alpha\Delta\vec{b}^{(i)}_u(t-1)$$

In a weight update pass, the changes are made to the reservoir of weights created prior to any parsing. The changes made are equivalent to summing up all of the weight changes in the parse tree on each weight layer in the reservoir. This is formalized in the below. We use the previously defined σ function to indicate a translation from OrCA node to grammar symbol. The σ^{-1} function is the inverse that translates a grammar symbol to the set of all of its occurrences in the NGN OrCA. This allows us to refer to the multiple weight changes distributed throughout the tree while summing them to their single allocation in the weight reservoir. The variable W is a weight object indexed by internal symbol $(x \in V)$. Note that this is done for clarity as we previously indexed W by production rules $(p \in P)$. Each weight layer as indexed here W_x is thus a concatenation of all of the weight matrices for

each production so that the total number of productions is q.

$$W_x = \left\langle \overset{|q|}{\underset{i=1}{C}} w_{\sigma(u \to v_i)}, \overset{|q|}{\underset{i=1}{C}} \vec{b}_{\sigma(u \to v_i)} \right\rangle$$

Finally, all of the components are ready and we can describe the weight update as follows.

$$W_x(t) = \left\{ \left\langle \sum_{i=0}^{\sigma^{-1}(x)} (\Delta w_{\sigma(i)}), \sum_{i=0}^{\sigma^{-1}(x)} (\Delta \vec{b}_{\sigma(i)}) \right\rangle \middle| x \in V \right\}$$

The result is a weight reservoir that is updated with the changes calculated from each instance for each weight and bias that occurs in the NGN.

3.10 Neural Grammar Network Generator Script

Writing the NGN parsing source is a mechanically regular and consistent task. Due to this regularity, a script was written in Python (van Rossum, 2008) to create the appropriate GNU bison, flex and C source files along with a make script that describes compilation of these components along with references to the in-house general neural network tree library. The generator script and its inputs are discussed next. A formal grammar is input to this generator along with some system parametres. This eases the process of deploying NGNs compatible with new versions of grammars as well as testing entirely new grammars. System parametres include the training constant (η) to use, the momentum coefficient (α) to use and with what magnitude to initialize the weights. The number of hidden units assigned to each node layer is also given here, mapped to the symbols of the grammar.

3.11 Chapter Conclusion

In this chapter, we visited the implementation and mathematical formalization of the NGN both structurally and functionally. The above description and methods sufficiently enable a

reader to replicate and extend the NGN technology. In the next chapter, we further reveal relevant details of computational chemistry and QSAR and how they pertain to the NGN.

Chapter 4

Application: Computational Chemistry

In this chapter, we describe the target application of computational chemistry. We discuss the problem of QSAR, representations of molecules in silico, the third party software used in this work as well as the work of other researchers in computational chemistry. We conclude this chapter with a trip back to the NGN in an example parse tree construction of an arbitrary three-carbon alcohol, isopentenol.

Computational chemistry is a method of research which leverages recent computing technology for high throughput chemical analysis. This arises as a natural integration of technology into the pure and physical sciences. Computational chemistry is a broad topic, so we cover only what is relevant for the function of and application problem of the NGN.

4.1 Cheminformatics

Cheminformatics is the accumulation of the approaches used to retain and process chemistry data in the context of information. Large cheminformatics projects are interested in

Table 4.1: A few example SMILES strings.

Molecule	SMILES String
1,3-Diphenyltetramethyldisiloxane	O([Si@@](c1ccccc1)(C)C)[Si@@](c1ccccc1)(C)C
1-Methoxy-4-(1-propenyl)benzene	C(=C/C)\c1ccc(cc1)OC
16b-OH-16-Methyl-3-methyl-estradiol	[C@@]12([C@H]([C@H]3[C@@H](c4c(CC3)cc (cc4)OC)CC2)C[C@@]([C@@H]1O)(O)C)C
17-Deoxyestradiol	[C@@H]12[C@@H]([C@H]3[C@](CC2)(CCC3)C) CCc2c1ccc(c2)O
17a-Estradiol	[C@]12([C@H]([C@H]3[C@@H](c4c(CC3)cc(cc4) O)CC1)CC[C@H]2O)C

maintaining databases, correlational statistics and data mining. In this thesis, we utilize two assets from cheminformatics, they are chemical markup languages and a suite of datasets.

4.1.1 Selected Languages SMILES and InChI

Efficient means to key large databases of chemical compounds have had the side effect of producing a number of string based chemical markups. Based on availability, two such markup languages have been selected. The first is called "Simplified Molecular Input Line Entry Specification" (SMILES)(James, 2007; Guha et al., 2006) and the second is called "IUPAC International Chemical Identifier" (InChI)(Stein et al., 2006) (where IUPAC stands for "International Union of Pure and Applied Chemistry"). Both of these languages attempt to capture all of the information relevant for a molecule while ensuring a single canonical representation. In general, SMILES and InChI strings do not flow over more than one line. Due to space limitations, long strings simply wrap around to the next line.

SMILES strings have a homogenous appearance and focus on describing atoms, the bonds between them, rings and branches. A few example SMILES strings can be found in Table 4.1. These molecules are selected from an Androgen Receptor Binding dataset (NCTR, 2007) (Fang et al., 2003).

InChI strings have a heterogeneous appearance where different layers describe different physical phenomena related to the molecule. These phenomena include the number of

Table 4.2: A few example InChI strings.

Molecule	InChI String
1,3-Diphenyltetramethyldisiloxane	InChI=1/C16H22OSi2/c1-18(2,15-11-7-5-8-12 -15)17-19(3,4)16-13-9-6-10-14-16/h5-14H,1-4H3
1-Methoxy-4-(1-propenyl)benzene	InChI=1/C10H12O/c1-3-4-9-5-7-10(11-2)8-6-9 /h3-8H,1-2H3/b4-3+
16b-OH-16-Methyl-3-methyl-estradiol	InChI=1/C20H28O3/c1-19-9-8-15-14-7-5-13 (23-3)10-12(14)4-6-16(15)17(19)11-20(2,22)18 (19)21/h5,7,10,15-18,21-22H,4,6,8-9,11H2,1-3H3 /t15-,16-,17+,18-,19+,20+/m1/s1
17-Deoxyestradiol	InChI=1/C18H24O/c1-18-9-2-3-17(18)16-6-4-12 -11-13(19)5-7-14(12)15(16)8-10-18/h5,7,11,15-17, 19H,2-4,6,8-10H2,1H3/t15-,16-,17-,18-/m0/s1
17a-Estradiol	InChI=1/C18H24O2/c1-18-9-8-14-13-5-3-12(19) 10-11(13)2-4-15(14)16(18)6-7-17(18)20/h3,5,10, 14-17, 19-20H,2,4,6-9H2,1H3/t14-,15-,16+,17-, 18+/m1/s1

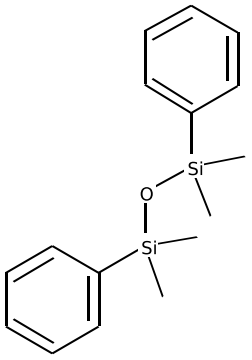
charges carried by particular atoms and the number of hydrogens bound at a given pH which may have added benefit in the QSAR problem space. Some example InChI strings can be found in Table 4.2. These are examples derived from the molecules expressed in the above SMILES.

Table 4.3 shows example molecules as molecular structure diagrams corresponding to the ones shown as SMILES and InChI strings before.

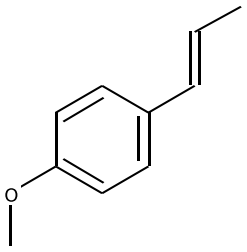
4.2 Quantitative Structure Activity Relationship Problems

Quantitative Structure Activity Relationship (QSAR) problems are a class of chemical modeling tasks. The objective is to produce a function which is capable of accepting as input a molecule and a returning as output a relevant real-value. The output is a value with some biochemical or medical significance. The function developed necessarily defines what kind of input is admissible. For many QSAR approaches, this input is a sequence of real-valued descriptors. Previous discussion has separated machine learning approaches

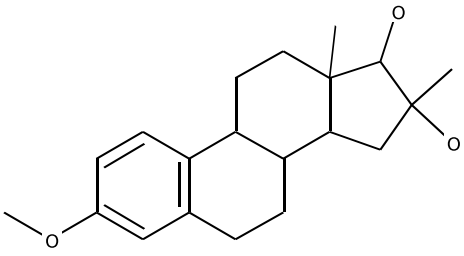
Table 4.3: A few example molecular structure (stick figure) diagrams.



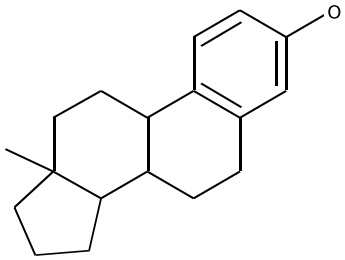
1,3-Diphenyltetramethyldisiloxane



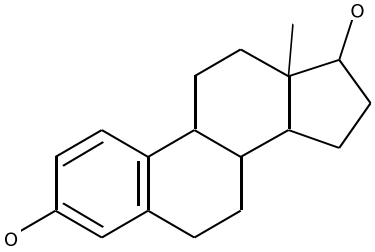
1-Methoxy-4-(1-propenyl)benzene



16b-OH-16-Methyl-3-methyl-estradiol



17-Deoxyestradiol



17a-Estradiol

used for QSAR into the traditional approaches and recent approaches (Sutherland et al., 2004). In traditional QSAR methods, suites of descriptors contain normalized real-valued transformations of such phenomena as melting point, gross molecular size, the counts of various atoms, functional groups and pharmacophores, the presence of aromatic bonds and rings, solubility etc., as well as some representations of transformed information about the 2D connection table. Some traditional suites of descriptors additionally incorporate hybrid 2D and 3D information dubbed '2.5D' descriptors. Traditional methods may also use well-established machine learning techniques instead of techniques designed specifically for QSAR. Such traditional machine learning approaches include Artificial Neural Networks (Karthikeyan et al., 2005) and Support Vector Machines (Jorissen and Gilson, 2005). For the NGN, we utilize the chemical string markups we described earlier as input instead of such chemical descriptors. A discussion of recent approaches follows.

It should be clarified that no bias should be given in favour of newer approaches without thorough investigation as performance varies from problem to problem.

4.2.1 Recent Machine Learning QSAR Approaches

QSAR approaches change as new machine learning devices become available, and as innovative metaphors for chemical representation are discovered. Next is overviewed recent machine-assisted QSAR approaches.

Recent QSAR approaches (Sutherland et al., 2004) define 3D methods for generating descriptors or revisit a novel 2D projection of a molecule. Four approaches of recent descriptor generation are discussed here, they are Comparative Molecular Field Analysis (CoMFA), Comparative Molecular Similarity Indices Analysis (CoMSIA), Anchor-GRid-INDependent (Fountaine et al., 2005) (Anchor-GRIND) and Holographic QSAR (HQSAR). CoMFA is a frequently used method describing 3D properties related to the potential energy of a binding between a pair of molecules. In CoMFA, a rectangular grid is used so that carbons aligned with vertices of a grid for a molecule contribute charges and steric (geometric strain) potentials to the grid. Different molecules are aligned to this grid and the distribution of charges

and steric potentials are compared in order to make a prediction about binding. CoMSIA is a related method which includes additional information about hydrogen bonds and hydrophobicity. Anchor-GRIND produces descriptors which relate the relative positions of atoms in a molecule to one another based on Molecular Interaction Fields. This is an improvement on the GRIND method by anchoring molecules against a particular common feature for all of the molecules in the dataset. Finally, HQSAR offers descriptions of the connections of a molecule by encoding the presence or absence of molecular fragments. This is an example of a revisitation to the projection of the 2D connection table for molecules.

Three example recent machine learning approaches developed specifically for handling QSAR are discussed next (Sutherland et al., 2003); they are Soft Independent Modeling by Class Analogy (SIMCA), Spline Fitting with a Genetic Algorithm (SFGA) and various decision tree topologies (Bruce et al., 2007). SIMCA calls for the creation of hypervolumes in the descriptor space so that datapoints of the same class occupy the same volume in much the same way as a SVM would divide classifications of points with hyperplanes. SFGA proceeds by fitting descriptors into a mathematical smoothing function (a spline) with a genetic algorithm. The importance of a descriptor and how it contributes to the final prediction of the model is modulated by evolving model parametres. A fixed number of generations is used to determine when to end training of the model. Various decision tree topologies have been used. A decision tree represents the elucidation of the class for a given molecule by a parsimonious selection of features. It has been noted that while this results in more humanly interpretable results, some predictive power is traded when compared with approaches that enclose such information. A collection of decision trees that nest properties differently can be aggregated together so that each tree contributes a vote to the overall decision. This kind of collection is called a decision forest (Tong et al., 2004).

4.2.2 Mathematical QSAR Approaches

One final interesting QSAR method is the application of a mathematical function approximation (Vighi et al., 2009). In this approach, molecules are each considered separately,

and the biological response curves are evaluated against the concentration of the molecule alone. Each molecule thus gets its own fitted mathematical equation. The mathematical regression is computer assisted but the final representation is a math expression along with a few parametres for that fitted expression. The fitted equation can thus be thought of as containing all of the relevant information required to make predictions about a given molecule. The drawback is that no predictions are possible about novel molecules of the same class without creating a new fitted equation based on biological or chemical assays.

4.2.3 Descriptor Dimensionality Reduction

Techniques which utilize descriptors benefit from dimensionality reduction in that vector units carrying redundant or irrelevant information are stripped from the suite of descriptors for a given problem. Furthermore, reducing the size of the input prevents the known problem of data overfitting. Problematic descriptors are items that contribute little novel information to the system in order to perform a classification or regression. Such items may not correlate with the expected output or may replicate another value in the system beyond a certain threshold for tolerance. Methods used to perform dimensionality reduction include Partial Least Squares (Sutherland et al., 2004) (PLS), Recursive Feature Elimination (Li et al., 2005) (RFE) and Principal Component Analysis (PCA) (Karthikeyan et al., 2005).

4.3 Discussion of Quantitative Structure Activity Relationship Problems

Inspiration based on present approaches and revisitation of them will guide the search for novel approaches with yet greater predictive performance in both learning machine and descriptor generation methods as well as a reduced cost in expert knowledge. In this vein, the NGN offers a descriptor-free method which is completely compatible with the novice chemist or computer scientist.

4.3.1 Clarification of QSAR-Like Bioinformatics Problems

Many other problems that resemble QSAR are addressed with bioinformatics. In these cases, biological sequence sensitivity and comparison are used rather than chemical descriptors. This tendency is a reflection of the way biological data is organized in nature versus chemical data. While some bioinformatics problems resemble molecular QSAR problems, they are not applicable to the NGN. Bioinformatics problems rely on machines that can perform on differences in sequence rather than the structured tree given by its syntax. Inducing the grammar of amino acids for instance is a task that has not yet been performed.

Before we conclude this chapter, we return to the NGN and illustrate representations of and an example parse for the molecule isopentenol.

4.4 An Example Molecule - Isopentenol

An arbitrary example molecule isopentenol has been selected to demonstrate the different in silico representations of molecules described above. This is by no means exhaustive, and is meant to give the reader an appreciation of the kinds of data used. An example parse given the SMILES and InChI syntax and grammar are shown as well. Isopentenol is chosen because it demonstrates branches, is small enough that parse trees for it are manageable and also for the presence of both single and double bonds.

4.4.1 Isopentenol as a Molecular Structure Stick Figure

Molecular structure diagrams also called stick figures are generally used to give an overview to a human reader of a molecule's shape, this is seen in Figure 4.1. Notice the double-bond, labeled hydroxyl "-OH" group and unlabeled carbons at each angle and branch. The unlabeled dangling ends of the molecule are hydrogens.

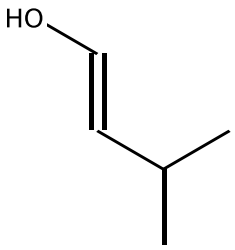


Figure 4.1: Isopentenol as a molecular structure diagram or stick figure.

4.4.2 Isopentenol as a SDF

The Structure Data File (Dalby et al., 1992) (SDF) is a molecular file format that will be explained in detail in the experimental section. Here is presented an example of what it looks like. The SDF file format offers a connection table of bonds as well as 2D or 3D coordinates for atoms about an arbitrary origin. The corresponding SDF for isopentenol is shown in Figure 4.2. Two sections of the SDF file are shown here; they are the 3D spatial coordinates and the connection table. The 3D spatial coordinates offers the location of each atom with respect to some arbitrary origin. The connection table indicates which atoms are connected together and with what kind of bond. In this example, '1' indicates a single bond, and '2' indicates a double bond. Throughout the file, the chemical symbols 'H', 'C' and 'O' stand for hydrogen, carbon and oxygen respectively. The additional entries to the right of both the spatial coordinates and the connection table are reserved for other functions not addressed in this paper. This SDF was found on PubChem, a free online molecular structure database (Wheeler et al., 2008).

4.4.3 Isopentenol as a SMILES String and NGN

Isopentenol is expressed in SMILES as "`CC(C)C=CO`". The corresponding NGN is shown in Figure 4.3. The left side of Figure 4.3 shows an even more simplified scheme of the NGN schematic. Weight layers are simplified to a single thin arrow, and activation layers are simplified to empty boxes. The right side shows an expansion for a subtree `chain` ← { `atom bond_chain` ← { `bond chain` }} in the familiar depiction used before. Note that alternative possible statements are not shown connected to the weight layers.

```
6440261
  -OEChem-04290922523D

 16 15  0     0  0  0  0  0  0999 V2000
    1.5882   -1.1184    0.5573 O   0  0  0  0  0  0  0  0  0  0  0  0
    1.3193    2.4019   -0.9075 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.1980    2.5988   -0.8431 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.8101    2.6121   -2.3420 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.8215    1.0737   -0.3885 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.0824    0.0665    0.0929 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.7764    3.1797   -0.2817 H   0  0  0  0  0  0  0  0  0  0  0  0
   -0.7272    1.8678   -1.4644 H   0  0  0  0  0  0  0  0  0  0  0  0
   -0.4716    3.5980   -1.1999 H   0  0  0  0  0  0  0  0  0  0  0  0
   -0.5653    2.5076    0.1849 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.5212    3.6025   -2.7104 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.9022    2.5439   -2.4008 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.3888    1.8630   -3.0220 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.9026    0.9429   -0.3994 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.0035    0.0822    0.1612 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.8310   -1.6600    0.8409 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  6  1  0  0  0  0
  1 16  1  0  0  0  0
  2  3  1  0  0  0  0
  2  4  1  0  0  0  0
  2  5  1  0  0  0  0
  2  7  1  0  0  0  0
  3  8  1  0  0  0  0
  3  9  1  0  0  0  0
  3 10  1  0  0  0  0
  4 11  1  0  0  0  0
  4 12  1  0  0  0  0
  4 13  1  0  0  0  0
  5  6  2  0  0  0  0
  5 14  1  0  0  0  0
  6 15  1  0  0  0  0
M  END
```

Figure 4.2: Isopentenol as a SDF.

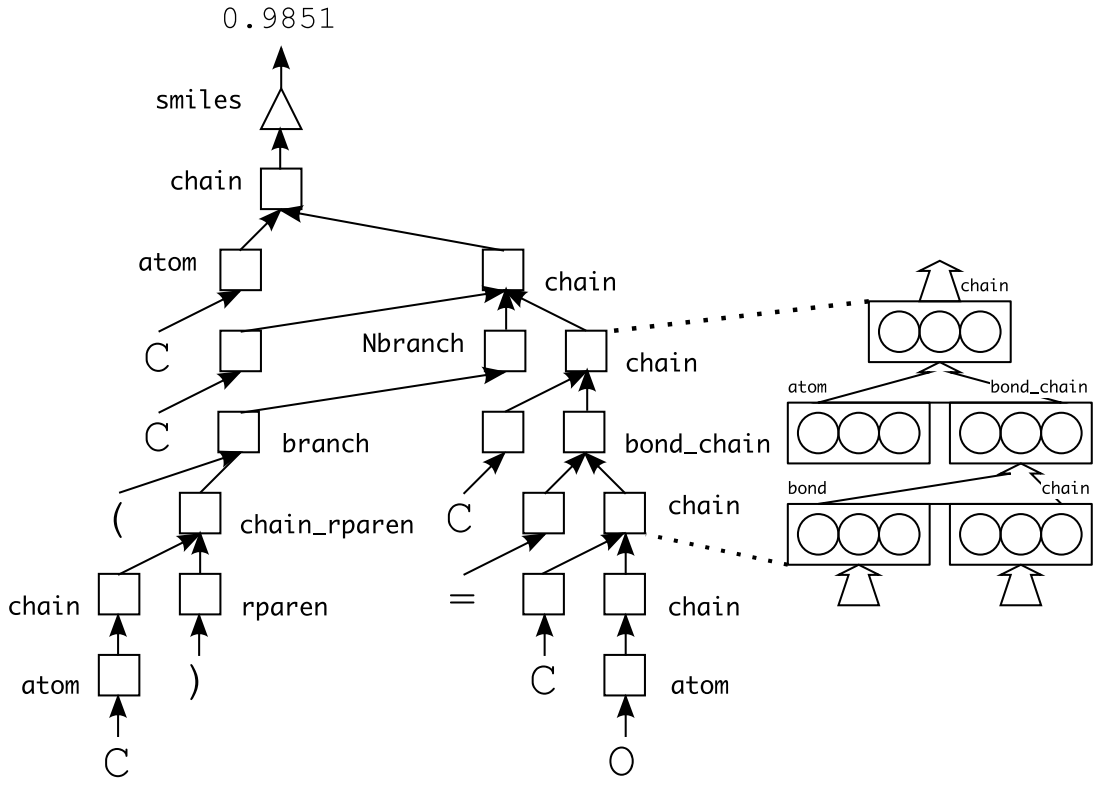


Figure 4.3: Isopentenol as a SMILES-NGN.

4.4.4 Isopentenol as an InChI String and NGN

Isopentenol is expressed in InChI as "`InChI=1S/C5H10O/c1-5(2)3-4-6/h3-6H,1-2H3/b4-3+`". Figure 4.4 shows the InChI-NGN structure for the backbone structure of the parse tree. InChI strings are organized in a series of conceptual layers; in this case, the layers are punctuated with symbols that start with "/". The subtrees corresponding to these layers are abbreviated as hexagons. These hexagons are expanded in Figure 4.5. This is done for visual clarity.

This backbone is the main structure that the layers of the InChI statement are attached to. This allows the InChI format to localize related kinds of data. The hexagons represent abbreviated subtrees corresponding to each InChI statement layer and are expanded in Figure 4.5. Notice that layers that occur in an InChI string are variable, and the above is only one possible selection.

The subtrees are attached to the InChI backbone structure. Each subtree corresponds to

a conceptual layer which conveys a different class of information. The "`Version`" subtree is metadata which identifies what version of syntax a given InChI string will obey; "`Formula`" provides a raw count of each atom type; "`C_body`" structures the connection table; "`H_body`" describes the position of hydrogens; "`B_body`" describes the positions of any double bonds. Notice that each layer and corresponding subtree have their own local grammar. This diagram is simplified, with intermediate layers removed.

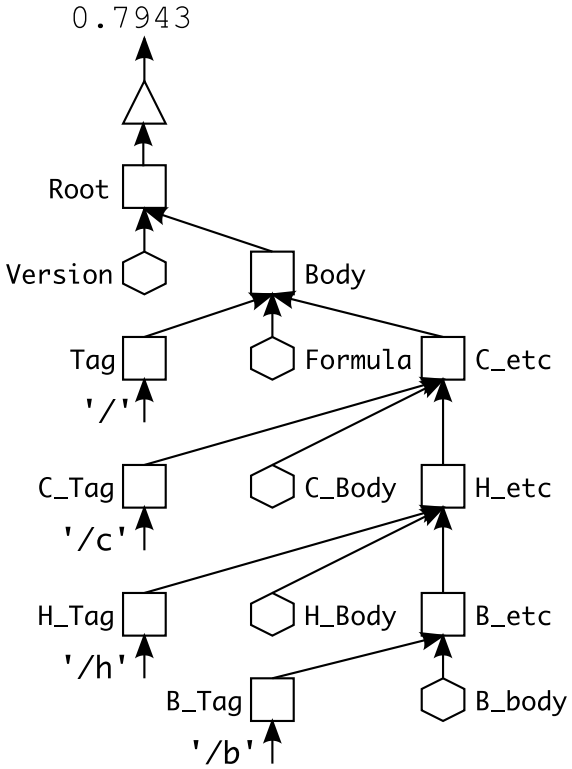


Figure 4.4: Isopentenol InChI-NGN backbone parse tree structure.

4.5 Chapter Conclusion

We have visited the problem domain of computational chemistry with particular focus on the problem of quantitative structure-activity relationship (QSAR). A discussion of technologies used to process and represent molecules in silico as well as the work of other researchers concerned with this problem has been completed. Finally, an example of these representations along with an NGN parse of these structures have been included. The reader now has

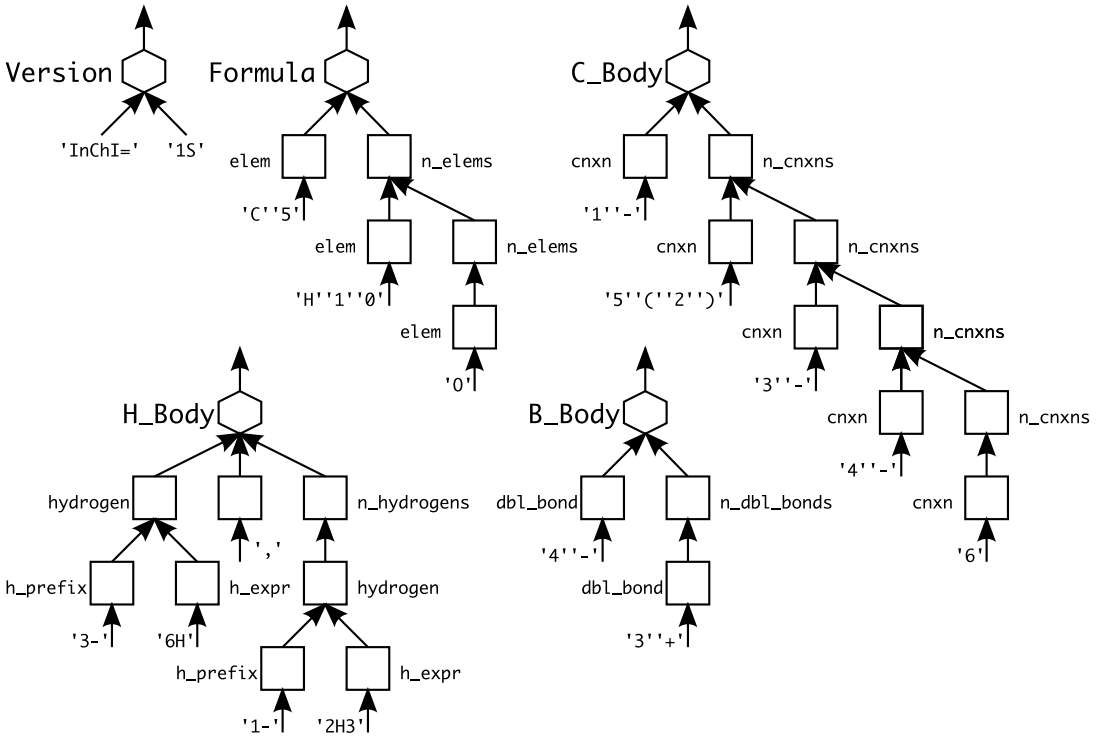


Figure 4.5: Isopentenol InChI-NGN layer parse tree substructures.

all of the information needed to understand the experimental section of this thesis, where we perform classification and regression tasks on a selection of QSAR data sets.

Chapter 5

Neural Grammar Networks in Quantitative Structure Activity Relationship

The NGN has been applied to some QSAR problems. Both classification and regression experiments have been conducted.

In this chapter, we first discuss the system parametres and experimental design used for each classification and regression tasks; we follow in the next chapter with results for each and finally conclusions that can be drawn.

5.1 General System Parametres

System parametres relevant for the discussion of the NGN revolve around concepts related to all artificial neural networks. In the upcoming mathematical explanation, it is revealed that the hidden layers of the NGN correspond to particular grammar symbols. The system may accept a different number of hidden units for each grammar symbol. The number of

hidden units, the training constant η, and momentum coefficient α are set differently depending on the task given to the NGN. These values are experimentally derived. Through experimentation, it was determined that larger random initial weights increased the probability of convergence of the NGN in a reasonable number of epochs (5000 for classification, and 7500 for regression). For the purposes of the NGN, weights are initialized with the option to define such a blindspot about 0.0 wherein no initial values may fall; the size of the blindspot is also determined experimentally.

5.2 General Experimental Design

Two experimental designs are run for each dataset where possible. The first is a Leave-X%-Out Cross Validation and the second mimics Designed Test Sets as described in other publications. Details for these designs are addressed separately in the following classification and regression sections.

Leave-X%-Out Cross Validation experiments follow the following design. A dataset of exemplars where all of the inputs and corresponding outputs are known is broken down into even groups of a given percentage. All of these subgroups are used as a training set and the remaining group is used as a test set. Each of these groups is used in turn as the test set and the performance is given as the average performance of the test system on each test set. Note that although the correct output is known to the user for all data points, the test set data points in each trial are unknown to the test system.

5.3 Dataset Selection

Datasets are selected on the criteria that they are freely accessible and that molecules expressed contain three dimensional conformations. The former allows for replicability and comparison to other work. The latter is important for QSAR methods that rely on structural descriptions. Three dimensional conformations enclose details such as Molecular

Table 5.1: A summary of the IC_{50} ranges in concentration and the threshold values in pIC_{50} for the datasets selected from Sutherland et al. (2003).

Dataset	Concentration Range IC_{50}	Threshold Value pIC_{50}
BZR	0.34nM to $> 70\mu$M	7.0
Cox2	1nM to $> 100\mu$M	6.5
DHFR	0.034nM to $> 1000\mu$M	6.0

bond rotations and angles that are important to the binding of molecules to target structures of particularly selective shapes.

An overview of the selected data proceeds. This overview is designed to address the biological relevance of each class of compounds being studied, the originating paper as well as the numeric transformation needed to scale and normalize the values for the NGN. A total of seven classification datasets are discussed followed by a total of ten regression datasets.

Three datasets are chosen from (Sutherland et al., 2003) for classification experiments. These are the Benzodiazepine Receptor Binders (**BZR**), the Cyclooxygenase-2 Inhibitors (**Cox2**) (Kauffman and Jurs, 2001), and the Dihydrofolate Reductase Inhibitors (**DHFR**). These datasets all convey the likelihood that a given molecule will bind to the named biological structures. The notions of 'binder' and 'inhibitor' are not thoroughly examined in this work, so that molecules that are strong binders may not necessarily be strong inhibitors. These two terms are left interchangeable for simplicity. The biological activity of these sets are described with IC_{50} (half maximal inhibitory concentration) values. An IC_{50} value is retrieved from an assay that determines the concentration of a molecule required to inhibit half of its biological activity. Because of the tendency for biological response curves to be hyperbolic, these values are converted to pIC_{50} where $pIC_{50} = -\log_{10}(IC_{50})$, so that molecules of higher potency have a higher pIC_{50} value (require exponentially less concentration to create the same desired effect). The threshold for classification is therefore guided by the distribution of data over the pIC_{50} value, and the threshold for classification as well. Table 5.1 summarizes the range of concentrations found in the dataset as well as the selected threshold.

The dataset corresponding to a Blood Brain Barrier (**BBB**) agent penetration study (Li et al., 2005) has been selected. The blood brain barrier is a defense which closes the brain off to many blood borne contaminants. A particle that can cross this barrier is said to be penetrating. For the preceding study and this work, an agent is said to be penetrating if at least 10% of its particles (by concentration) have crossed over to the brain side after a steady state has been achieved.

A Factor Xa Inhibitor (**FXa**) dataset from (Fountaine et al., 2005) is selected. Binding affinity was measured with K_i, the reaction constant which indicates how likely a molecule-inhibitor pair is to form. When $K_i > 1\mu M$, the compound is said to have low affinity (is a non-binder). When $K_i \leq 10nM$, the compound has high affinity (binder).

Finally, two datasets were retrieved for classification from the Endocrine Disruptor Knowledge Base, a feature of the National Center for Toxicological Research (NCTR, 2007). These datasets relate the binding affinities of a set of Estrogen Receptor Binders (Blair et al., 2000)(Branham et al., 2002) (**ER**) and a set of Androgen Receptor Binders (Fang et al., 2003) (**AR**). The tendency to bind is given as a Relative Binding Affinity (RBA) score. Due to the hyperbolic nature of this data, the tenth base logarithm (logRBA) is used. The range of ER data in logRBA is 3.56 to 2.27, and of AR data in logRBA is -4.50 to 2.60 where the threshold is the lower value in those ranges (non-binders are denoted by an arbitrary high-negative value, -5×10^3 or -10^4).

For regression, nine datasets were chosen from (Sutherland et al., 2004). These sets are Angiotensin Converting Enzyme Inhibitors (**ACE**); datasets of **BZR**, **Cox2** and **DHFR** different than the ones used in the classification studies above; Thermolysin Inhibitors (**Therm**); Thrombin Inhibitors (**Thr**) (Bohm et al., 1999); and Glycogen Phosphorylase B Inhibitors (**GPB**) (Gohlke and Klebe, 2002). The **ACE** and **Therm** datasets were gathered in previous work by Depriest et al. (1993). The real-valued ranges and units used to express binding affinity are described in Table 5.2 where pIC_{50} represents binding affinity as a value inversely proportional to the concentration needed for a given effect and where pK_i relates to the probability of two molecules binding together.

Table 5.2: A summary of the real-valued ranges of activity for datasets seen in Sutherland et al. (2004).

Dataset	Activity Range	Unit of Measure
ACE	2.1 to 9.9	pIC_{50}
Cox2	4.0 to 9.0	pIC_{50}
Therm	0.5 to 10.2	pK_i
Thr	4.4 to 8.5	pK_i
AChE	4.3 to 9.5	pIC_{50}
GPB	1.3 to 6.8	pK_i

The final two datasets used for the regression task is the **ER** and **AR** sets described earlier. All of the non-binders are stripped away for the regression task, so that only binders which have legal values assigned to them are used. The regression in this case is solely a determination of how well a known binder behaves, rather than a task that potentially includes labeling how poorly a non-binder behaves as well.

5.3.1 Normalization

For the classification task, molecules that are positive classifiers are tagged with the value "0.8" while values that were non-classifiers had value "0.2". Similarly, after the logarithmic transformations necessary, molecules used in regression tasks were linearly normalized to the scale [0.2, 0.8]. These numbers are chosen for for the internal logistics transfer function of the NGN. Earlier experiments which simply normalized values to [0.0, 1.0] ended up with a poor convergence probability, or with poorer generalization capability likely due to problem overfitting. Training is made possible with output values within the asymptotic boundaries of $\{0.0^+, 1.0^-\}$.

5.4 Structured Data File

Structure Data File (Dalby et al., 1992) (SDF) format is the format used by most of the dataset repositories to express molecules. This format describes three major items about a

molecule. First, a set of coordinates addresses the locations of each atom relative to some arbitrary origin. Coordinates always occur in ordered triplets, each element referring to a particular spatial dimension. Many datasets have molecules represented as 2D projections. This is done with a mathematical transformation that reduces the third entry in the triplet to zero. Second, a connection table indicates which atoms are connected and also the kind of bond in that connection. Third, additional information defined by the user can be used to tag the molecules. This tagged information may include real values produced from biological or chemical assays as well as metadata such as the molecule's origin. Note that the arbitrary absolute coordinate system can used as a reference to produce a rotational and translational invariant representation such as the SMILES and InChI formats.

5.5 OpenBabel

OpenBabel (OpenBabel, 2008) is an open source software package used to convert SDFs into SMILES and InChI for this work. This is always done with the chiral options enabled for SMILES and isomeric options enabled in InChI. These options preserve useful 3D conformation information. The algorithm used to generate these strings are canonical, meaning that the molecule is always traversed the same way in the representation. This determinism is also important for repeated trials and reproducibility.

5.6 Classification Experiments

Classification experiments are discussed first. This section commences with an explanation about the design and parametres used. We follow with an overview of the math functions used to elucidate the performance for this task.

Table 5.3: A summary of the final experimental parametres used in the Leave-20%-Out classification experiments and designed training set versus test set classification experiments.

Variable	Variable Denotation	Value
η	Training Constant	0.60
α	Momentum Coefficient	0.90
RMSE	Root Mean Squared Error Convergence Threshold	0.05
	Initial Random Weight Value Range	[-1.6, -1.0] $\cup$ [1.0, 1.6]
	Maximum Number of Epochs Before Restarting	5000
	Number of Hidden Nodes used by SMILES-NGN	8
	Number of Hidden Nodes used by InChI-NGN	8
	Output Scale	(0.2, 0.8)

5.6.1 Classification Experimental Design

For classification, Leave-20%-Out experiments proceed by creating five subsets from a given dataset. Each subset is used in turn as the test set while the remainder is used for training. For repeated experimental trials, each of these subsets were left out ten times each, so that fifty trials are conducted in total and each exemplar is in an untrained test case in ten trials. To compare against previous work wherein authors have prescribed fixed test and training sets, the same sets are used as defined and run for a total of ten trials; this is also referred to as designed test and training sets respectively. Summarized in Table 5.3 are the system parametres used for the classification experiments. These parametres were experimentally derived.

Summarized in Table 5.4 are some properties of the datasets used. Molecules in each dataset act as a binder or inhibitor on the named biological structure (except for the **BBB** set which describes molecules' ability to transit through the blood brain barrier). N^+ is the number of positively classifying data points and N^- is the number of negatively classifying points.

Table 5.4: A summary of the datasets used in classification experiments.

Dataset	Dataset Full Name	Size	N^+	N^-	Reference
BZR	Benzodiazepine Receptor	405	230	175	Sutherland et al. (2003)
Cox2	Cyclooxygenase 2	467	273	194	Sutherland et al. (2003) Kauffman and Jurs (2001)
DHFR	Dihydrofolate Reductase	756	302	454	Sutherland et al. (2003)
BBB	Blood Brain Barrier	415	276	139	Li et al. (2005)
FXa	Factor Xa	435	279	156	Fountaine et al. (2005)
ER	Estrogen Receptor	232	131	101	NCTR (2007) Branham et al. (2002) Blair et al. (2000)
AR	Androgen Receptor	202	146	56	NCTR (2007) Fang et al. (2003)

5.6.2 Classification Experimental Evaluation

Four measures are often used to describe the performance of classification methods, they are accuracy (Q), sensitivity (SE), specificity (SP) and Matthew's Correlation Coefficient (MCC). These scores are defined below such that 'TP' is a count of true positives, 'TN' is a count of true negatives, 'FP' false positives and 'FN' false negatives.

$$Q = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

$$\mathrm{SE} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$\mathrm{SP} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

$$\mathrm{MCC} = \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FN} \cdot \mathrm{FP}}{\sqrt{(\mathrm{TP} + \mathrm{FN})(\mathrm{TP} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})}}$$

In each of the functions above, a high score indicates some measure of better performance over a low score. The use of SE and SP separately evaluate the performance of a test classification system for correct positive and correct negative classification respectively. Notice that MCC is the only value that attempts to compensate for an unbalanced amount of correct positive classification versus correct negative classification in one single value.

Next, we visit the parametres and design used in regression task experiments.

5.7 Regression Experiments

An exploration into the regression capabilities of the NGN is conducted as well in QSAR problems. We utilize again a cross validation design followed by designed test and training sets as per previous work. We end on an overview of functions which allow us to describe how well the test system performs.

5.7.1 Regression Experimental Design

Experimental parametres specific to the regression task are discussed here. A Leave-5%-Out design was used. Twenty groups of approximately equal size were created out of each dataset used. Each group was in turn used as the test set while the remainder was used as the training set. NGN network parametres were systematically and experimentally derived. Each parameter was varied independently of the rest until a combination of reasonable performance was achieved. These parametres are described in Table 5.5. To compare against methods performed in other papers, designed test sets are used that consist of one third of the dataset. Parametres used for such designed datasets are similarly derived and are summarized in Table 5.6

The sizes of the regression datasets are summarized in Table 5.7. The molecules corresponding to each dataset either act on the named biological structure as a binder or an inhibitor.

The distribution of data points for regression datasets are shown in the figures in Tables 5.8 to 5.10. These tables are vertical bar graphs so that the domain is represented by ten even segments and that the domain is normalized from zero to one. The range is the proportion of data points which falls into each of the ten segments. Notice that many of these datasets are biased so that there are more entries in in some domain segments than

Table 5.5: A summary of the final experimental parametres used in the Leave-5%-Out regression experiments.

Variable	Variable Denotation	Value
η	Training Constant	0.30
α	Momentum Coefficient	0.10
RMSE	Root Mean Squared Error Convergence Threshold	0.03
	Initial Random Weight Value Range	[-1.6, -1.0] $\cup$ [1.0, 1.6]
	Maximum Number of Epochs Before Restarting	7500
	Number of Hidden Nodes used by SMILES-NGN	8
	Number of Hidden Nodes used by InChI-NGN	8
	Output Scale	[0.2, 0.8]

Table 5.6: A summary of the final experimental parametres used in the designed regression experiments. One third of the data is used as a test set, while the remaining is used for training.

Variable	Variable Denotation	Value
η	Training Constant	0.33
α	Momentum Coefficient	0.66
RMSE	Root Mean Squared Error Convergence Threshold	0.04
	Initial Random Weight Value Range	[-1.2, -0.4] $\cup$ [0.4, 1.2]
	Maximum Number of Epochs Before Restarting	7500
	Number of Hidden Nodes used by SMILES-NGN	8
	Number of Hidden Nodes used by InChI-NGN	8
	Output Scale	[0.2, 0.8]

Table 5.7: A summary of the datasets used in regression experiments.

Dataset	Dataset Full Name	Size	Reference
ACE	Angiotensin Converting Enzyme	114	Sutherland et al. (2004) Depriest et al. (1993)
Cox2	Cyclooxygenase-2	282	Sutherland et al. (2004)
Therm	Thermolysin	76	Sutherland et al. (2004) Depriest et al. (1993)
Thr	Thrombin	88	Sutherland et al. (2004) Bohm et al. (1999)
AChE	Acetylcholinesterase	111	Sutherland et al. (2004)
GPB	Glycogen Phosphorylase B	66	Sutherland et al. (2004) Gohlke and Klebe (2002)
BZR	Benzodiazepine Receptor	163	Sutherland et al. (2004)
DHFR	Dihydrofolate Reductase	397	Sutherland et al. (2004)
AR	Androgen Receptor	146	Fang et al. (2003)
ER	Estrogen Receptor	131	Blair et al. (2000) Branham et al. (2002)

others. These figures are included to show the variable distributions of data between the different sets used for the regression task.

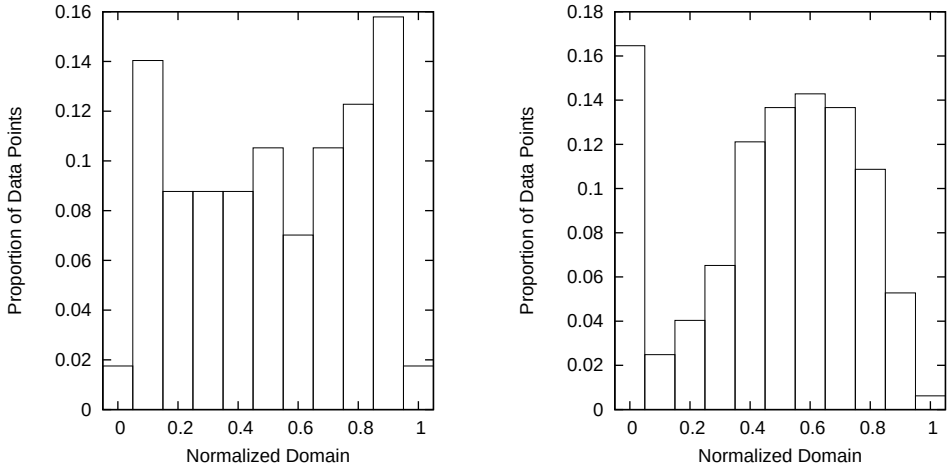
5.7.2 Regression Experimental Evaluation

A conventional measure of predictive performance used in QSAR is r^2_{pred} (predicted r-squared). This is defined as follows where i is the index of a data point, y is the expected output value, "prediction" is the function being tested, x is the molecule used as input and SD is the standard deviation of the test dataset.

$$\text{PRESS} = \sum_{i=1}^{N_{test}} (y_i - \text{prediction}(x_i))^2$$

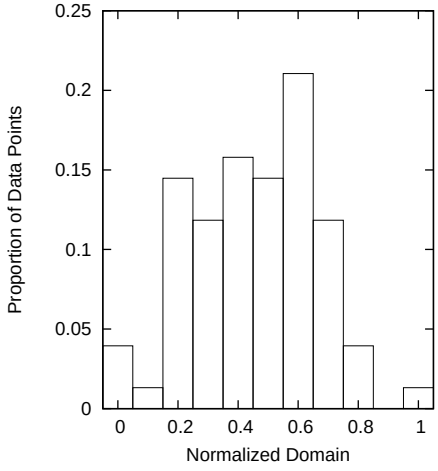
$$\text{SD} = \sum_{i=1}^{N_{\text{test}}} (y_i - \bar{y}_{\text{test}})^2$$

Table 5.8: Distribution of datapoints for regression datasets ACE, Cox2, Therm and Thr.

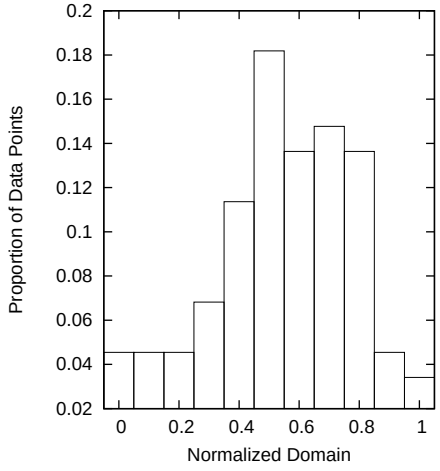


ACE

Cox2

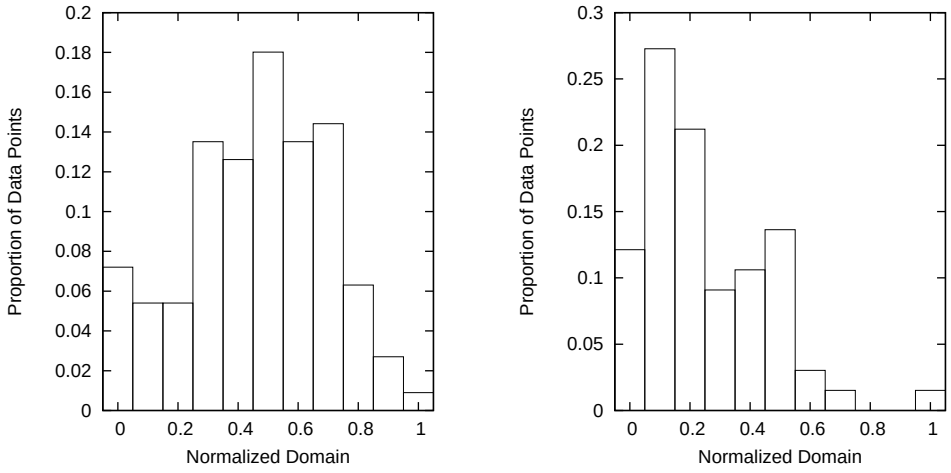


Therm



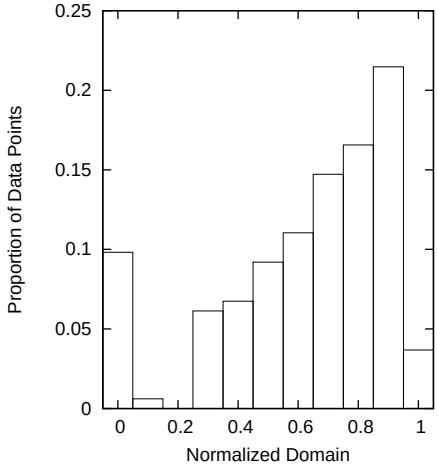
Thr

Table 5.9: Distribution of datapoints for regression datasets AChE, GPB, BZR and DHFR.

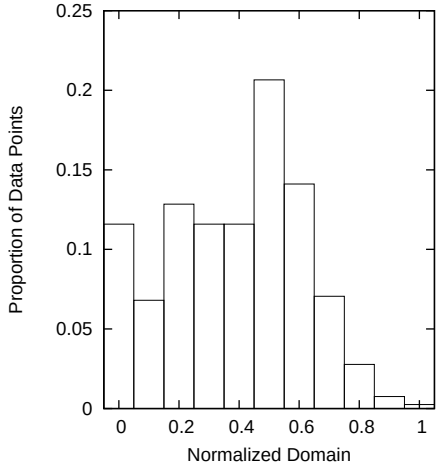


AChE

GPB

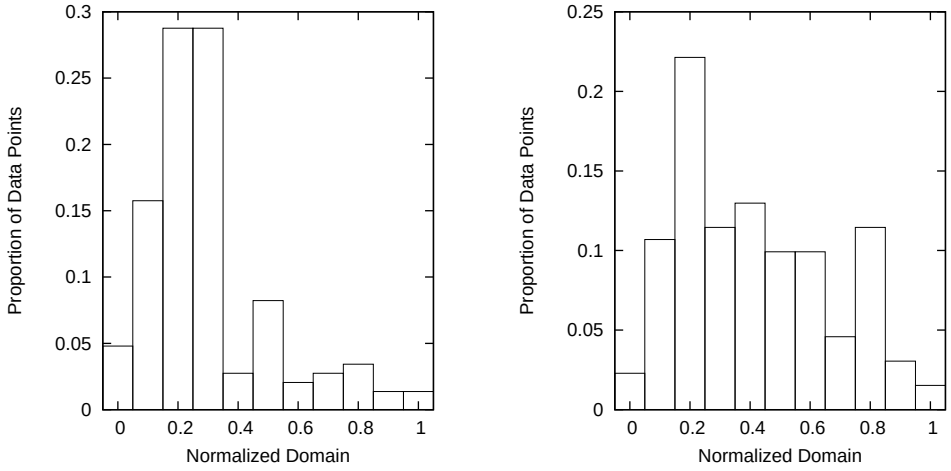


BZR



DHFR

Table 5.10: Distribution of datapoints for regression datasets AR and ER.



AR

ER

$$r^2_{\text{pred}} = \frac{\text{SD - PRESS}}{\text{SD}}$$

The PRESS value (predictive residual sum of squares) and SD value are not generally used as a measure of predictive accuracy on their own, rather they are stepping stones to achieve the r^2_{pred} value. A higher r^2_{pred} score is indicative of better performance. This is mathematically congruent to indicating that there is less deviation from the curve that represents the correct distribution of target values.

5.8 Chapter Conclusion

In this chapter, we have addressed experimental design for each classification and regression as well as how performance is judged by various math functions. Following this chapter is a discussion of the results obtained from the experimental designs mentioned.

Chapter 6

Experimental Results for NGN in QSAR

In this chapter, we discuss the experimental results for each the classification and regression tasks for the selected datasets as well as compare system performance against existing devices. We conclude with a discussion of the benefits of utilizing the NGN.

6.1 Classification Experimental Results

The classification results are organized as follows. The first half consists of a comparison of the NGN against other published methods and their performance. This is presented as a suite of tables comparing the NGN against the work of five prior studies and internal cross validation design. Where available, values are presented with their standard deviations as indicated after the plus-minus symbol ($\pm$). The second half consists of a set of nine graphs describing the Leave-20%-Out Cross Validation design and is used to illustrate the internal validity of the NGN on these datasets.

The first three tables (Table 6.1 to Table 6.3) relate the performance on the datasets discussed by Sutherland et al. (2003) (**BZR**, **Cox2** and **DHFR**). In keeping with compar-

Table 6.1: A summary of the performance on the **BZR** dataset with results as reported by Sutherland et al. (2003) compared to the InChI-NGN in this work.

Design	Method	Q(%)	SE(%)	SP(%)	MCC
40% Test Set	SIMCA	72	68	76	-
	RP	69	64	74	-
	SFGA	75.5	70	81	-
	InChI-NGN	63.2	62.1	64.6	0.265
Leave-20%-Out	SIMCA	71.5±11.0	73±10	70±12	-
	RP	65.5±12	68±12	65±12	-
	SFGA	68.5±12	69±11	68±13	-
	InChI-NGN	69.9±1.98	73.4±1.87	65.3±2.29	0.387±0.041

ing against methods used previously, two designs are used. First, a designed 'maximum dissimilarity' testing set is used consisting of 40% of the data. The remaining data composes the training set. The maximum dissimilarity algorithm is a heuristic which selects a reasonably diverse group of molecules from the dataset. The intention of this design is to test the generalization capability of a trained classification model. Second, the standard Leave-20%-Out CV design applies. The methods compared are Soft Independent Modeling by Class Analogy (SIMCA), Recursive Partitioning (RP) and Spline-Fitting with a Genetic Algorithm (SFGA).

For these three first experiments, the NGN performance is quite variable compared to the methods introduced by Sutherland et al. (2003). The NGN performs best at the Leave-20%-Out CV design for **DHFR** compared to other methods, and overall does best at the **DHFR** dataset, also achieving second place in the designed test set. The **DHFR** dataset is the largest set (756 entries) but does not have an even distribution of data points (roughly 2 positive for every 3 negative classifiers). The size of the dataset may be sufficient reason for this good performance. In the remaining datasets **BZR** and **Cox2**, the NGN performs at last and third last place respectively for both CV and designed test sets. These two datasets are average size, with no particular overpowering bias toward positive or negative classifiers. The performance on the **Cox2** CV experiment is redeemable however, achieving over 70% classification accuracy.

Table 6.2: A summary of the performance on the **Cox2** dataset with results as reported by Sutherland et al. (2003) compared to the InChI-NGN in this work.

Design	Method	Q(%)	SE(%)	SP(%)	MCC
40% Test Set	SIMCA	71	75	67	-
	RP	71	79	63	-
	SFGA	73.5	75	72	-
	InChI-NGN	65.1	62.5	68.4	0.307
Leave-20%-Out	SIMCA	78±9	79±9	77±9	
	RP	69.5±12	72±12	67±12	-
	SFGA	74±9.5	76±9	72±10	-
	InChI-NGN	72.2±1.36	74.4±1.01	68.7±2.45	0.421±0.275

Table 6.3: A summary of the performance on the **DHFR** dataset with results as reported by Sutherland et al. (2003) compared to the InChI-NGN in this work.

Design	Method	Q(%)	SE(%)	SP(%)	MCC
40% Test Set	SIMCA	75.5	74	71	-
	RP	65	57	73	-
	SFGA	68.5	71	66	-
	InChI-NGN	73.2	73.1	100.0	0.029
Leave-20%-Out	SIMCA	63.5±9.5	57±10	70±9	-
	RP	61±12	57±12	65±12	-
	SFGA	64.5±10.5	65±11.0	64±10.0	-
	InChI-NGN	74.8±1.63	70.3±2.44	77.5±1.72	0.471±0.035

Table 6.4: A summary of predictive scores for the **BBB** dataset as presented in the work by Li et al. (2005) followed by the performance of the InChI-NGN.

Method	Q(%)	SE(%)	SP(%)	MCC
LR	57.1	63.6	42.8	0.063
LDA	46.8	40.0	58.4	-0.067
C4.5 DT	73.8	83.7	54.9	0.398
k-NN	70.8	77.0	58.0	0.348
PNN	70.3	76.2	57.8	0.357
SVM	71.0±4.53	89.9±3.16	64.3±13.07	0.524±0.117
LR RFE	71.0	83.9	46.4	0.321
LDA RFE	71.2	78.2	58.3	0.360
C4.5 DT RFE	74.3	80.3	62.8	0.433
k-NN RFE	77.1	85.5	61.4	0.477
PNN RFE	76.1	84.3	62.1	0.481
SVM RFE	83.7±3.90	88.6±7.01	75.0±12.83	0.645±0.080
InChI-NGN	72.0±2.33	77.6±1.72	59.0±3.91	0.355±0.052

A comparison of the classification performance on the **BBB** dataset is found in Table 6.4. This table compares the InChI-NGN against methods tried by Li et al. (2005). These methods are Logistic Regression (LR), Linear Discriminate Analysis (LDA), Decision Tree (C4.5 DT), k-Nearest Neighbor (k-NN), Probabilistic Neural Network (PNN) and Support Vector Machine (SVM). The use of a molecular descriptor refinement procedure called Recursive Feature Elimination is indicated by the suffix 'RFE'. Data is consistently treated with Leave-20%-Out (or 5-Fold) Cross Validation trials in order to keep comparisons valid. In this comparison, the InChI-NGN performs better than any method without RFE, but is fifth out of seven approaches when RFE is introduced. Because the NGN does not use molecular descriptors, it cannot benefit from RFE.

Table 6.5 presents findings for the experiments done with the **FXa** dataset compared against methods presented by Fountaine et al. (2005) and Mohr et al. (2008). These methods are Anchor-Molecular Interaction Fields (A-MIF), MIF-MIF and two Molecule Kernels (MK1 and MK2) in P-SVMs. Experiments are run with a designed test set consisting of $^{\mathbf{1}}/_{\mathbf{3}}$ of the available data with the remainder used for training. An additional cross validation was done that is not reflected in the original study. The **FXa** dataset represents a task

Table 6.5: A comparison of the work done by Fountaine et al. (2005) and Mohr et al. (2008) against the InChI-NGN for the **FXa** dataset.

Design	Method	$Q(\%)$	SE(%)	SP(%)	MCC
$^{\mathbf{2}}/_{\mathbf{3}}$ Train, $^{\mathbf{1}}/_{\mathbf{3}}$ Test	A-MIF	88	-	-	-
	MIF-MIF	84	-	-	-
	MK1	94.5	98.7	89.5	-
	MK2	95.2	98.9	87.7	-
	InChI-NGN	83.8	84.3	82.9	0.657
Leave-20%-Out	InChI-NGN	86.4±2.50	88.5±0.87	82.7±0.05	0.705±0.052

that was particularly easy for the NGN, accomplishing 83.8% concordance for the designed test set and 86.4% for five fold (20%) CV. The NGN comes in last place compared to other methods, but is compatible with the results obtained by Fountaine et al. (2005). Mohr et al. (2008) has the best performing results overall.

Two additional tables are included for classification experiments performed on the **ER** (Table 6.6) and **AR** (Table 6.7) datasets retrieved from a compilation from NCTR (2007). Presented are values useful for internal validity of the NGN. The **ER** experimental results are displayed along with results presented by Tong et al. (2004). Notice that the results of these experiments cannot be directly compared because the internal cross validation was split differently. While we used a 20% CV, Tong et al. (2004) used a 10% CV who employed a decision forest model. No comparison is possible with the **AR** dataset as there are no such QSAR classification experiments done in the literature for **AR**. The decision forest method may offer better performance than the NGN for the explicit ability to describe relevant features in a tree while the NGN must automatically discover and internally express any such relationships. This may be particularly important for estrogen receptor binders. The performance on the **AR** dataset is compatible with many recent methods, exceeding a concordance of over 75%.

A summary of the performance of nine Leave-20%-Out Cross Validation classification experiments on combinations of grammars and datasets are included (Figure 6.1 to Figure 6.5). The graphs describe network performance as follows. Networks are trained and

Table 6.6: Summary of the Decision Forest performance Tong et al. (2004) against the NGN performance on **ER**.

Design	Method	$Q(\%)$	SE(%)	SP(%)	MCC
Leave-10%-Out	Decision Forest	81.9	-	-	-
Leave-20%-Out	SMILES-NGN	69.3±2.28	71.7±2.39	66.0±2.74	0.373±0.047
	InChI-NGN	66.1±1.70	69.5±2.65	61.6±1.33	0.309±0.040

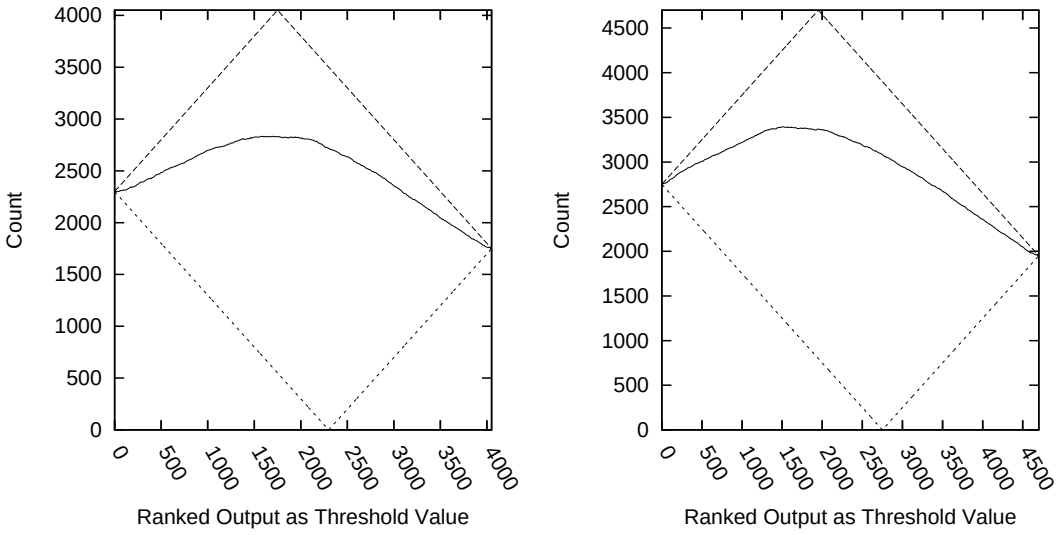
Table 6.7: Summary of the NGN performance on **AR**.

Design	Method	$Q(\%)$	SE(%)	SP(%)	MCC
Leave-20%-Out	SMILES-NGN	70.3±0.91	71.8±0.46	37.5±11.4	0.052±0.033
	InChI-NGN	76.3±3.20	80.7±2.86	60.6±7.02	0.380±0.098

their response values to input strings recorded. These response values are sorted. A threshold value in a classification experiment determines whether a prediction should be considered a positive prediction or a negative prediction. These sorted response values are each used as the threshold value in turn. All of the remaining values are then compared to this threshold, and a count of the correct predictions is taken. Because the distribution of values is not uniform, each threshold should be considered a class. Each threshold class is mapped across the x-axis, and a count of correct classifications on that threshold is the height of the curve at that point on the y-axis. The enclosing parallelogram indicates the range of possible response curves that may exist given this visualization scheme. When the curve is considered as a whole, the closer the curve is to the upper boundary of the parallelogram, the better it has performed. In practice, the threshold class which produces the highest point on the response curve corresponds to the most predictive threshold value for the dataset from which the predictive model was trained. This threshold value would ideally be the place on the domain on which one would find the best value for accuracy (concordance, Q), sensitivity (SE), specificity (SP) and Matthew's correlation coefficient (MCC).

The nine graphs correspond to combinations of grammars and datasets that successfully converged. They hence represent the entire range of performance ever exhibited by the NGN

for classification.



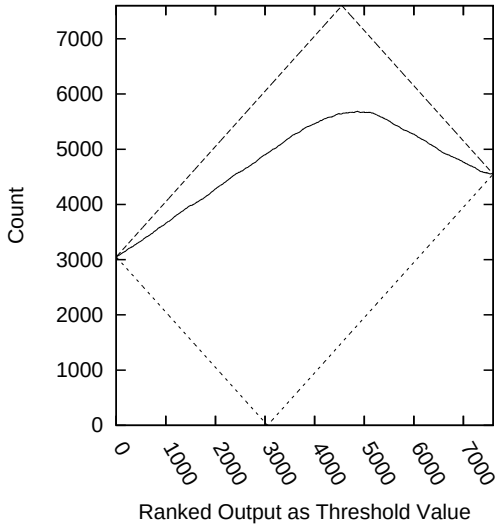
(a) BZR 20% Cross Validation - InChI-NGN (b) Cox2 20% Cross Validation - InChI-NGN

Figure 6.1: A summary of performance on two classification experiments, InChI-NGN on benzodiazepine receptor inhibitors and cyclooxygenase-2 inhibitors.

6.2 Classification Experiment Discussion

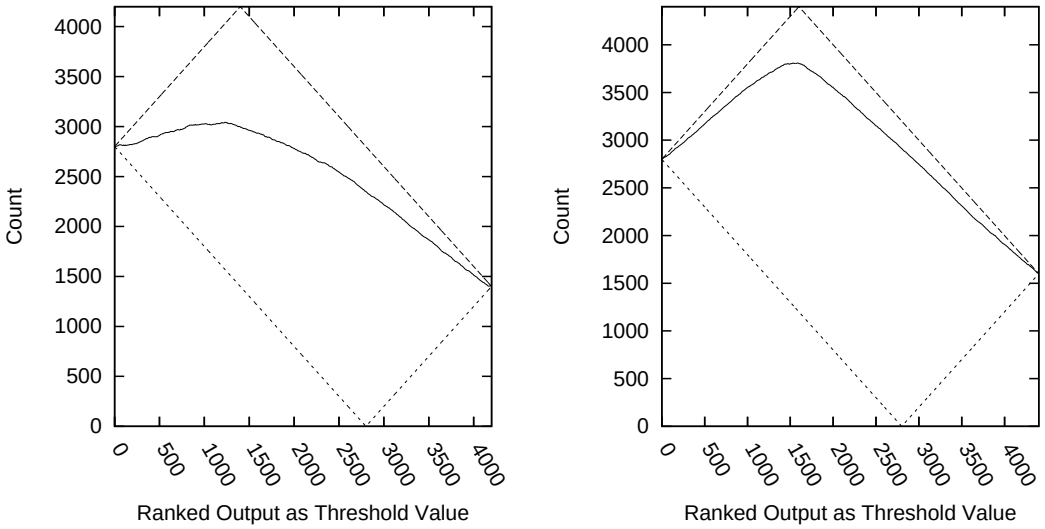
The most convincing performance on visual inspection occurs at the results of the **FXa** dataset on InChI-NGN in Figure 6.3b. Indeed, this corresponds to the best numeric performance of 86.4±2.50%. The **FXa** dataset selection performed by Fountaine et al. (2005) is designed by selecting molecules with a common substructure called the benzamidine moiety. In the originating study, this moiety is used as a common anchoring point in a 3D modeling scheme. No other data used in this work were specifically designed with anchoring substructures in mind which lends thought that the NGN is better suited for this sort of problem. A comparison with a broader range of methods on the same and similar anchor-sensitive data could clarify the role of the NGN in this potential.

The experiments that are indicated with the most difficulty are the **BZR** and **Cox2** datasets on the InChI-NGN for the designed test sets described by Sutherland et al. (2003). These performed with accuracies 63.2% and 65.1% respectively falling short of the next



(a) DHFR 20% Cross Validation - InChI-NGN

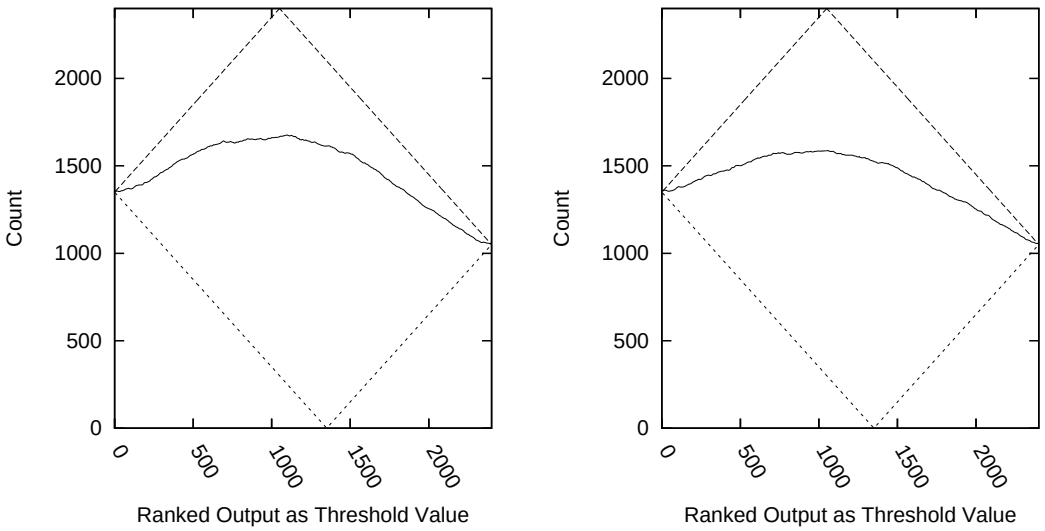
Figure 6.2: A summary of performance on a classification experiment, InChI-NGN on dihydrofolate reductase inhibitors.



(a) BBB 20% Cross Validation - InChI-NGN

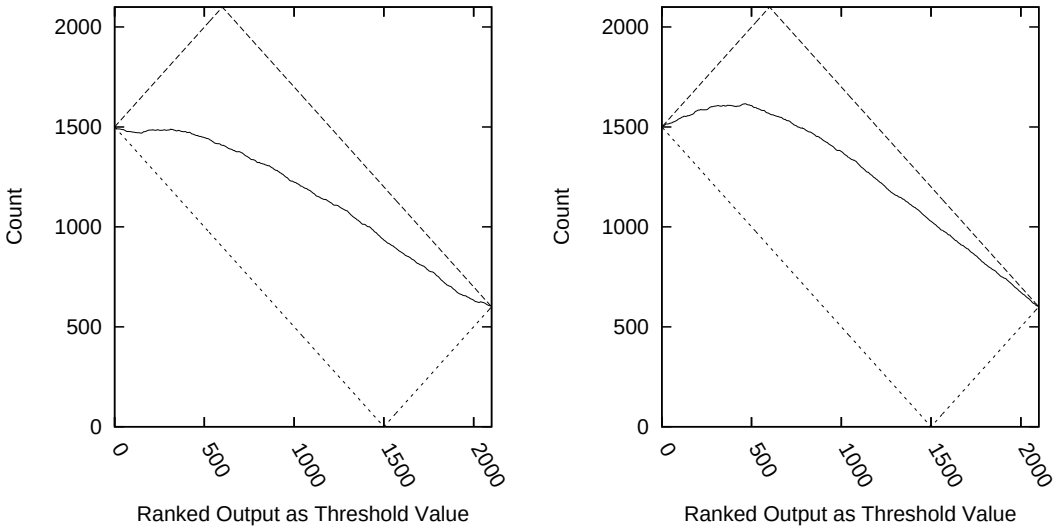
(b) FXa 20% Cross Validation - InChI-NGN

Figure 6.3: A summary of performance on two classification experiments, InChI-NGN on blood brain barrier penetrating particles and factor Xa inhibitors.



(a) Estrogen 20% Cross Validation - SMILES-NGN (b) Estrogen 20% Cross Validation - InChI-NGN

Figure 6.4: A summary of performance on two classification experiments, SMILES-NGN and InChI-NGN on estrogen receptor binders.



(a) Androgen 20% Cross Validation - SMILES-NGN (b) Androgen 20% Cross Validation - InChI-NGN

Figure 6.5: A summary of performance on two classification experiments, SMILES-NGN and InChI-NGN on androgen receptor binders.

worst method compared by 6% both times. The **DHFR** dataset discussed in the same originating study, however performed well with 73.2% accuracy on the designed test set and scored within 3% of the top compared method. The values describing the data are all negative-log-normalized values which pertains to the same biological characteristic of inhibiting a target molecule. In each case, the overall data set is selected for representing a diverse range of molecules while the test set is a subset of maximum dissimilarity. This is in contrast with the **FXa** dataset which is selected so that each molecule contains some common anchoring feature. The final major difference in performance may be attributed to the absolute size of each dataset, where **BZR**, **Cox2** and **DHFR** consist of 405, 467 and 756 molecules respectively. This trend should be tested and if valid would suggest that a greater absolute number of training exemplars facilitates extrapolation into highly varied test sets. Interestingly, wherever variance was recorded and compared between methods, the NGN always had a much smaller value which indicates a high model stability.

6.3 Regression Experimental Results

An internal cross validation experiment is done along with an experiment to compare the performance of the NGN against previous work in the literature. It is discovered that the performance of the NGN is better for the designed data sets over internal cross validation in terms of average performance and model stability. In general, the reliability of performance may become an issue. In practice however, these concerns can be addressed convincingly.

The experimental results of the the Leave-5%-Out trials are shown in Table 6.8. Each 5% segment was left out three times, yielding a total of sixty experimental trials. The experiments are sorted by descending performance. Standard deviations are also listed after the plus-minus ($\pm$) symbol. The BZR and DHFR datasets did not converge for cross-validation tests.

Regression performance scores r^2_{pred} are included in Table 6.9 which compares the performance of the NGN on the designed datasets. Table 6.9 is sorted on descending per-

Table 6.8: The r^2_{pred} scores for converging Leave-5%-Out Cross Validation regression experiments.

Grammar	Dataset	Dataset Full Name	r^2_{pred}
SMILES	**ACE**	Angiotensin Converting Enzyme	0.386±29.20
	AR	Androgen Receptor	0.382±18.29
	ER	Estrogen Receptor	0.349±21.82
	GPB	Glycogen Phosphorylase B	0.253±43.18
	AChE	Acetylcholinesterase	0.193±35.26
	Therm	Thermolysin	-0.288±210.30
InChI	**ER**	Estrogen Receptor	0.476±11.16
	ACE	Angiotensin Converting Enzyme	0.383±15.09
	Cox2	Cyclooxygenase-2	0.279±8.97
	Therm	Thermolysin	0.247±25.96
	AR	Androgen Receptor	0.119±79.50
	Thr	Thrombin	0.088±99.79
	AChE	Acetylcholinesterase	0.061±39.70
	GPB	Glycogen Phosphorylase B	-0.180±104.87

formance for SMILES-NGN. Methods employing descriptor generation (Sutherland et al., 2004) include Comparative Field Analysis (CoMFA), Eigenvalue Analysis (EVA), Holographic QSAR (HQSAR) and Traditional 2.5D Descriptors (2.5D). Methods employing a graph representation of molecules include Molecule Kernel 1 (MK1) and Molecule Kernel 2 (MK2) used in a Support Vector Machine approach (Mohr et al., 2008). Values missing from MK1 and MK2 were experiments not performed for the originating paper. These figures are included for context. Datasets are designed so that one third of the data is in the training set and the remainder is in the test set. For the NGN, ten trials are committed for each dataset for each grammar; standard deviations are indicated after the plus-minus (±) symbol.

In addition to the results shown in Table 6.9, some results for specific Leave-5%-Out CV experiments were carried out by Shi et al. (2001). These results are shown in Table 6.10. The QSAR methods are the already discussed CoMFA and HQSAR. In these experiments the NGNs on average have placed last, but have a very wide spread in performance. A future task seeing the use of a validation set design could be used to increase the selection of potentially better NGNs prior to continuing onto tests on the test sets; this kind of design

Table 6.9: A summary of r^2_{pred} scores for the NGN compared to other methods on datasets described for regression in this work.

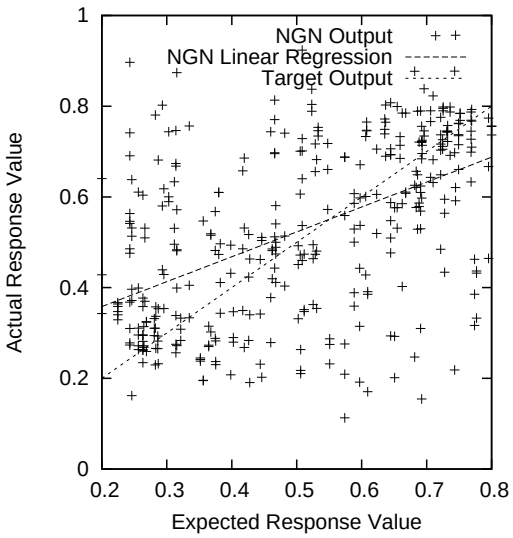
Dataset	SMILES-NGN	InChI-NGN	CoMFA	EVA	HQSAR	2.5D	MK1	MK2
GPB	0.79±0.23	0.48±2.90	0.42	0.49	0.58	0.04	—	—
ACE	0.74±0.46	0.78±0.31	0.49	0.36	0.30	0.51	0.58	0.55
AChE	0.68±0.80	0.60±0.78	0.47	0.28	0.37	0.16	0.50	0.48
Cox2	0.56±1.33	0.37±4.28	0.29	0.17	0.27	0.27	—	—
Therm	0.47±2.72	0.52±1.59	0.54	0.36	0.53	0.07	—	—
BZR	-0.29±17.30	0.11±8.74	0.00	0.16	0.17	0.20	0.34	0.36
Thr	—	0.70±0.72	0.63	0.11	-0.25	0.28	—	—
DHFR	—	0.66±0.94	0.59	0.57	0.63	0.49	0.64	0.65

Table 6.10: Regression results for Leave-5%-Out CV for the techniques CoMFA and HQSAR as performed in Shi et al. (2001) versus the NGN.

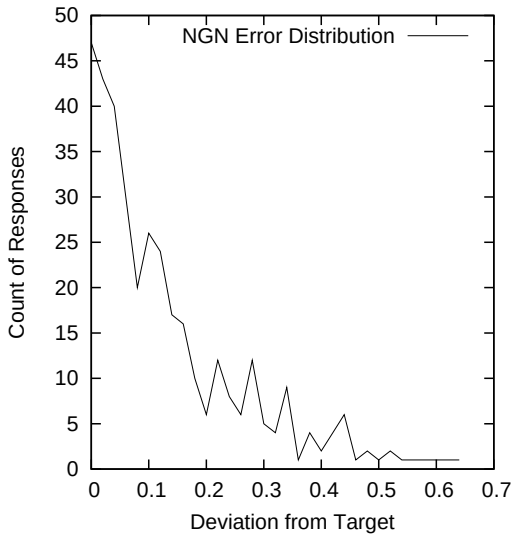
Design	Method	r^2_{pred}
Leave-5%-Out	CoMFA	0.652±0.0126
	HQSAR	0.585±0.0189
	SMILES-NGN	0.349±21.82
	InChI-NGN	0.476±11.16

is expanded in detail later in this chapter.

Included are performance graphs for Leave-5%-Out experiments on select grammar and dataset combinations (Figure 6.6 to Figure 6.13). These graphs are chosen for demonstrating the best eight predictive r^2_{pred} scores out of fourteen experiments that successfully converged. Graphs are shown in pairs. Graphs on the left side are indicated as follows. NGN outputs are plotted against the desired output from the test set. A line running across the major diagonal would represent perfect performance. Included on the graphs are a linear regression line, and such a line of perfect performance. Graphs on the right side show the distribution of error so that the domain is the absolute difference between a given data point as output by the test system and the target output value; this is counted on the range for all data points for every trial.

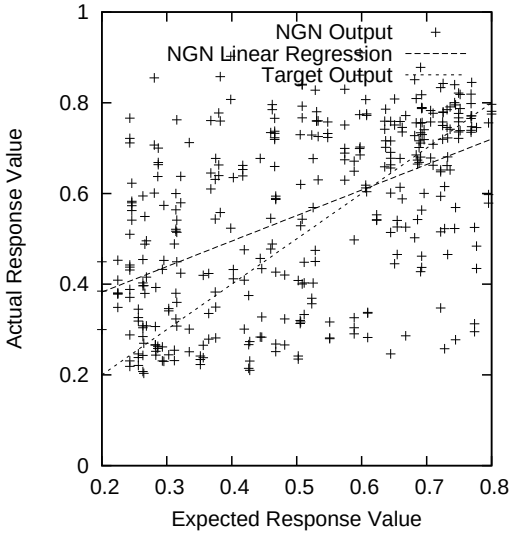


(a) ACE 5% Cross Validation - SMILES-NGN

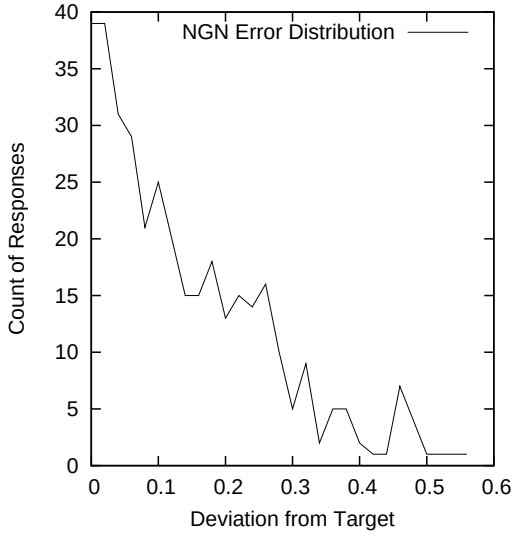


(b) ACE 5% Cross Validation - SMILES-NGN

Figure 6.6: A summary of performance on the 5% Cross Validation regression experiment on SMILES-NGN for angiotensin converting enzyme inhibitor dataset.

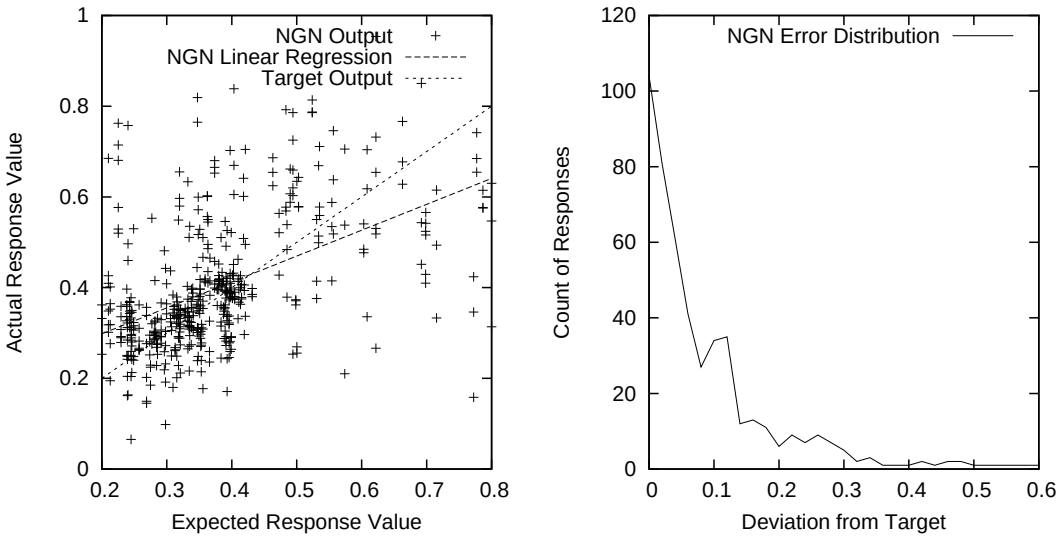


(a) ACE 5% CV Regression Line - InChI-NGN



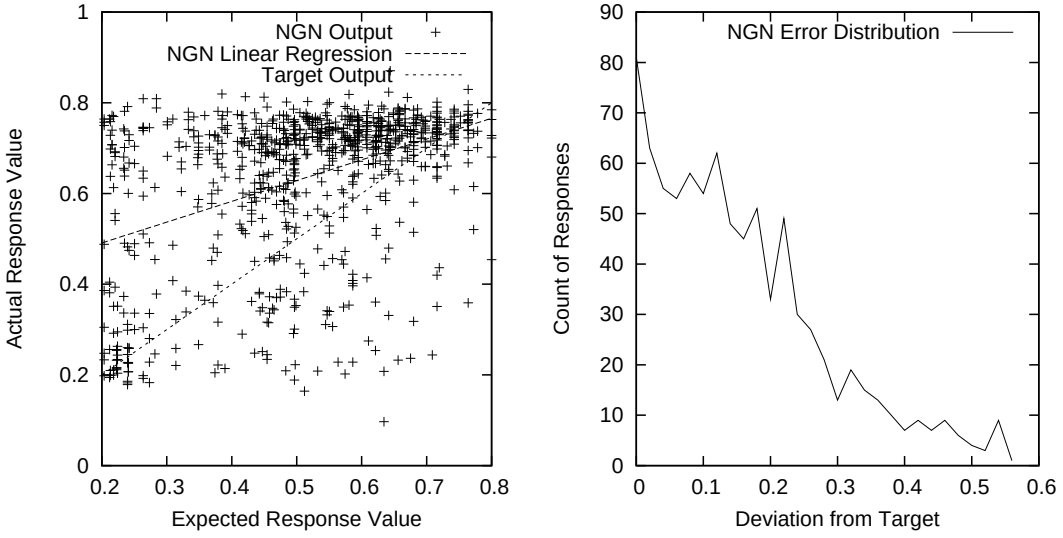
(b) ACE 5% CV Residuals - InChI-NGN

Figure 6.7: A summary of performance on the 5% Cross Validation regression experiment on InChI-NGN for angiotensin converting enzyme inhibitor dataset.



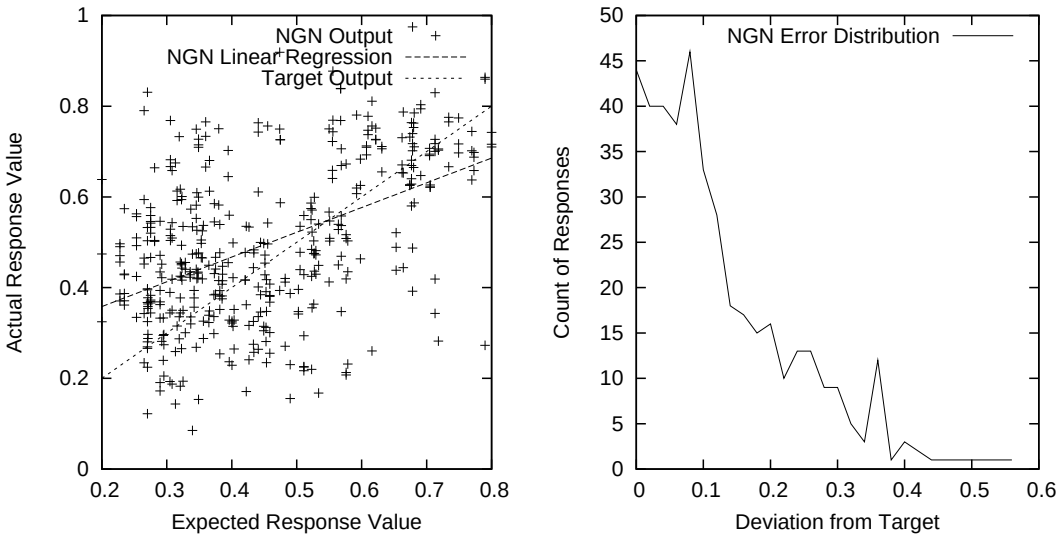
(a) Androgen 5% Cross Validation - SMILES-NGN(b) Androgen 5% Cross Validation - SMILES-NGN

Figure 6.8: A summary of performance on the 5% Cross Validation regression experiment on SMILES-NGN for androgen receptor binder dataset.



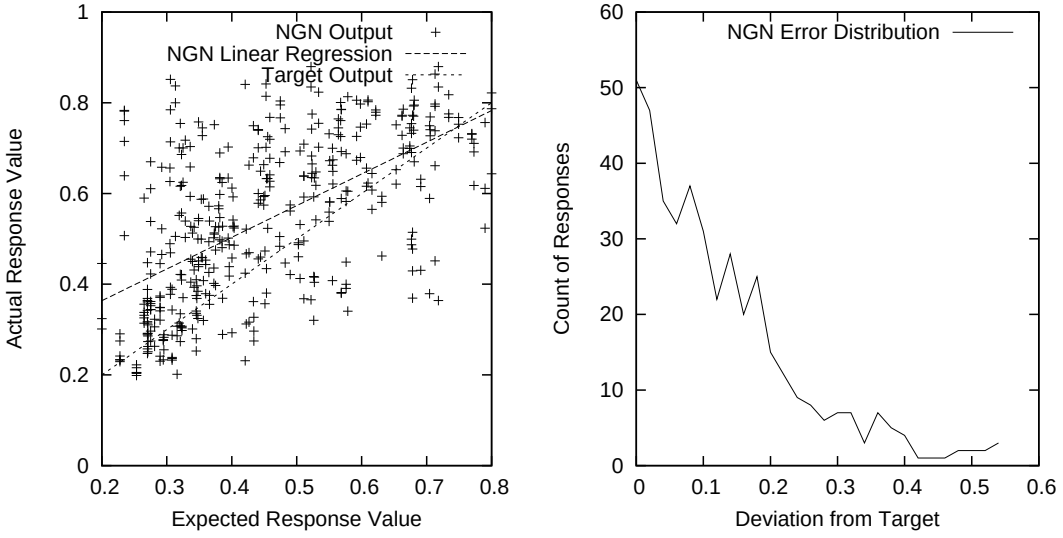
(a) Cox2 5% Cross Validation - InChI-NGN (b) Cox2 5% Cross Validation - InChI-NGN

Figure 6.9: A summary of performance on the 5% Cross Validation regression experiments on InChI-NGN for cyclooxygenase-2 inhibitor dataset.



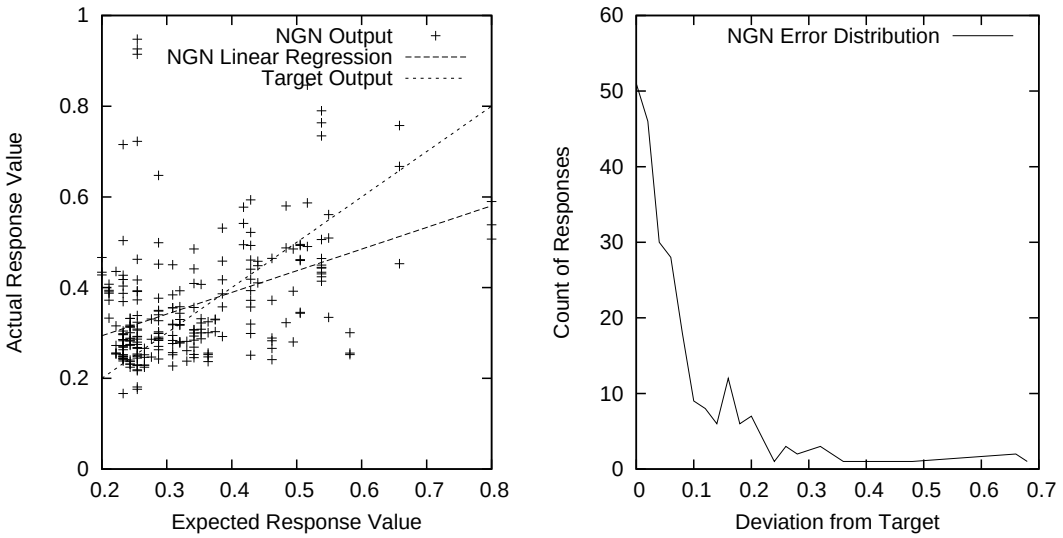
(a) Estrogen 5% Cross Validation - SMILES-NGN (b) Estrogen 5% Cross Validation - SMILES-NGN

Figure 6.10: A summary of performance on the 5% Cross Validation regression experiments on SMILES-NGN for estrogen receptor binder dataset.



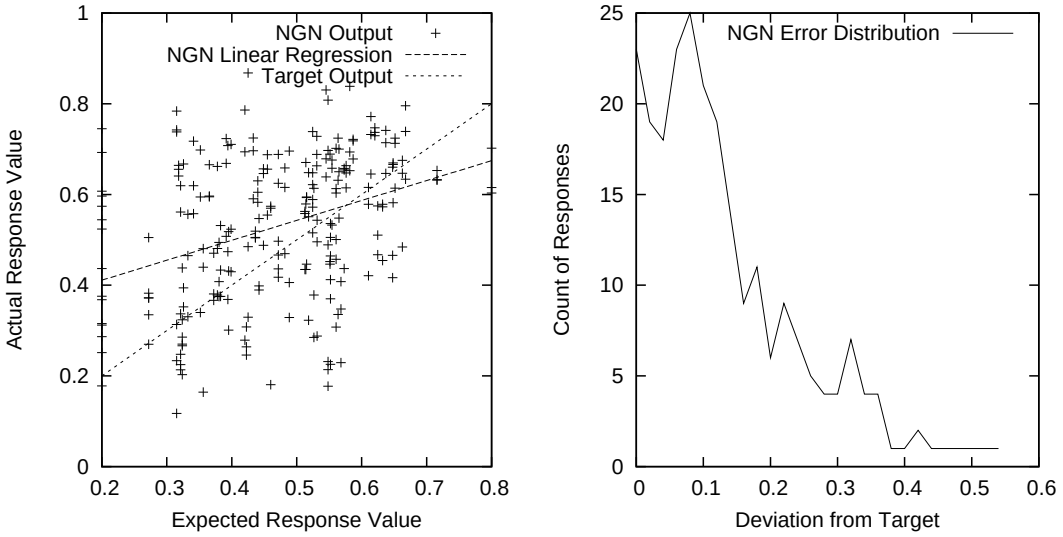
(a) Estrogen 5% Cross Validation - InChI-NGN (b) Estrogen 5% Cross Validation - InChI-NGN

Figure 6.11: A summary of performance on the 5% Cross Validation regression experiments on InChI-NGN for estrogen receptor binder dataset.



(a) GPB 5% Cross Validation - SMILES-NGN (b) GPB 5% Cross Validation - SMILES-NGN

Figure 6.12: A summary of performance for the 5% Cross Validation regression experiments on SMILES-NGN for glycogen phosphorylase B inhibitor dataset.



(a) Therm 5% Cross Validation - InChI-NGN (b) Therm 5% Cross Validation - InChI-NGN

Figure 6.13: A summary of performance for the 5% Cross Validation regression experiments on InChI-NGN for thermolysin inhibitor dataset.

6.4 Regression Experiment Discussion

One of the problems for the NGN in regression was a wide spread of performance. This problem manifests itself by an occurrence of a variety of r^2_{pred} scores between trials whereas in an ideal situation, a correlation for prediction should remain the same. This became problematic in the initial evaluation of the NGN because on average, relatively good r^2_{pred} scores were accomplished. Upon breaking down the average into its constituent trials, the spread became obvious. Representative experiments are shown in Figure 6.14 which demonstrate the best and worst case spread. Ten trials are shown each in total to illustrate the spread in performance. The r^2_{pred} scores are sorted in ascending order. The best case variance and performance in spread is demonstrated by **GPB** regression experiments while worst case is demonstrated by **BZR**.

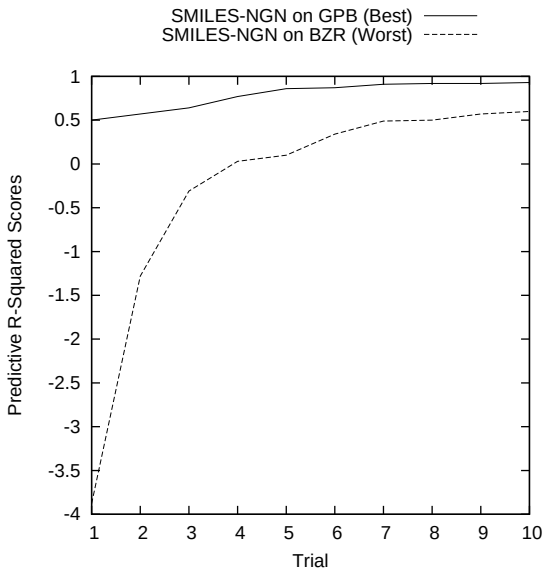


Figure 6.14: Two representative cases for the performance of the NGN are shown for the designed test sets where one third of the data points are used as a test set.

A future experiment could deal with the spread problem by introducing a validation set. The large spread may contain trials with some very good scores. A validation set can be used to separate these good models out from a collection of assorted performance. A validation set could be used to test the generalization ability of a trained NGN. If the performance

generates a r^2_{pred} score below an acceptable threshold, the NGN is rejected. Successful models are then used to test the designed test set. Depending on how representative the validation set is, this may be a sufficient way to improve the reliability of models generated with the NGN. To ensure that the scores are still compatible for comparison, the validation set must be constructed from the training set so that the same test set used in other methods can still be used with the new model.

The best performance obtained by the NGN was performed by the SMILES-NGN on the **GPB** dataset. This is a particularly small dataset consisting of only 66 data points. The size of the dataset alone is not sufficient to explain the performance as the next smallest sets **Thr** and **Therm** of size 88 and 76 respectively are not next in line for performance. The worst performance is also accomplished by SMILES-NGN. This occurs for the **BZR** dataset and exhibits a negative score for prediction (only noise was learned). Interestingly, **BZR** is also the most difficult for the InChI-NGN, and near the bottom for performance for the remaining methods. This dataset consists of 163 data points and is neither particularly large or small, so that there must be something about the biological property of these molecules that is particularly difficult. Finally, the best performance achieved by InChI-NGN is on the **ACE** dataset. Besides being the second easiest dataset for SMILES-NGN, there is nothing particularly remarkable about **ACE** that could explain the performance. It is possible that the relevant physical attributes for the regression of this set is particularly exposed in the SMILES and InChI representations.

6.5 Chapter Conclusion

In this chapter we have shown the compatibility of the NGN in QSAR classification problems with respect to other recent techniques and demonstrate its potential to regression tasks although model stability remains a task to be further resolved. The NGN demonstrates a spread of competence depending on the kind of chemical problem it is presented with; the precise nature of this spread remains to be elucidated and formalized. Coupled with the

potential for yet better training techniques and the independence from expert knowledge, the NGN offers the potential of a future viable alternative to present QSAR solving devices.

Chapter 7

Conclusion

In this thesis, the Neural Grammar Network was revealed. This novel machine learning system has demonstrated capacity for classification and potential for regression in the Quantitative Structure-Activity Relationship problem space of Computational Chemistry. Implementation details and experimental findings have been reported for replicability and a mathematical formalization has been described for completion. Natural extensions to present QSAR work as well as to the NGN at large have been indicated. Future directions follow in the chapter below.

Chapter 8

Future Work

There is much work left to be done with the NGN. Here are some natural extensions that can be explored with the NGN.

8.1 Future Work in Neural Grammar Network-QSAR Problems

QSAR problems represented the field of application for this thesis. The NGN has shown comparable performance to other machine learning approaches for classification and shows promise in future work for regression problems. Optimizing the NGN for QSAR problems can involve many items. There may be particular kinds of chemical problems that are better suited for this system. Those problems that rely on classification by the presence or absence of close proximity token elements such as functional groups or a short series of aromatic bonds in a ring structure are well suited. Problems that rely on the gross structure of the molecule, and the determination of mass, size or volume of a molecule would likely do poorly. This claim is made due to the way data is treated in the NGN. Approximating function given differences in molecular size requires the NGN to become sequence sensitive, capable of learning the distances between two relevant tokens. This is a task that an ordinary

recurrent framework cannot accomplish. Recognizing features expressed by tokens in close proximity however, is solvable.

Aside from selecting specific cases of QSAR problems, there are many parametres that can be experimentally derived. The training constant, momentum coefficient, number of hidden nodes in a layer, the value of initial weights and the gross formal grammar structure can all be edited. If it's discovered that a certain token combination is important yet linearly inseparable, inserting a dummy grammar symbol to break up the tokens would fix that problem. If the depth of the network is too great and the error propagation signal is decaying too much, opportunistically merging and deleting irrelevant or problematic grammar symbols would solve that.

Finally, other formal string based expressions for molecules could be used. This includes implementing and deploying new chemical markups, or re-expressing the molecules in a non-canonical but still mathematically consistent way.

8.1.1 Canonicalization Rules and String Permutation

Exploring the many permutations of a string that represent a molecule may yield useful insight about whether certain expressions are more useful to the NGN for different QSAR problems, and by extension future problem spaces. The NGN is liable to exhibit the same limitations as other recursive architectures; namely that error propagation in the gradient descent algorithm exponentially decays as it traverses down the network (Hochreiter et al., 2001). A permutation which places tokens of greater relevance to the top of the parse tree may be more useful to the NGN. Two drawbacks exist for the exploration of permutation that will have to be addressed. First, since the generation of each permutation can be considered a graph traversal, the number of strings that can be generated grows exponentially with the number of atoms in a molecule. Second, care must be taken to ensure that the NGN does not learn to label permutations based on properties of the strings which accidentally capture information outside of what is actually available from the data. Both of these problems can be solved by designing a suite of canonicalization algorithms, so that

each algorithm deterministically yields the same permutation of a molecule each time it is applied. This way, the explosion of permutations is avoided and a fair treatment is given to only data that actually exists in the molecule.

8.2 Future Work in Neural Grammar Network Training and Architecture

A general exploration on practical improvements addressing individual known issues with the NGN performance is discussed here.

It is impossible to evaluate the number of hidden units to assign to a given node layer without some detailed mathematical heuristic which analyzes the number of occurrences of a given node layer based on the grammar and the training set of strings. In fact, doing such an indepth analysis would offset any advantage of the automated nature of the NGN. It is possible however to apply a genetic algorithm to discover a useful combination of hidden units for a specific problem and grammar. An array of integers so that each integer corresponds to the number of hidden units of a single grammar symbol would map the chromosome for this genetic algorithm. The stocasticity of the genetic algorithm would permit the use of only small random subsets of data in generational training and generalization tests. Such a system would also see potential in parallelization.

One limitation that has been mentioned about recurrent architectures in general is the inability to cope with sequence recognition. Sequence recognition relates to distinguishing patterns of symbols, lengths of those patterns and the distances between two relevant tokens. A solution which can be implemented in the layers of a neural network is the long short-term memory (Hochreiter and Schmidhuber, 1997) (LSTM) cell which essentially operates as a logical latch in a neural network. Recursive architectures that have used LSTM cells in the past have solved the sequence sensitive problems described above.

These future experiments may prove to be worthwhile as they address specific shortcomings of the present NGN.

8.3 Characteristics of Future Application Spaces

Future application spaces must have data that can be defined by a formal grammar. With the current architecture, the function approximation tasks must be based on syntax and not sequence sensitivity so that a combination of the syntax and input tokens alone can capture all of the needed information in a classification or regression task. Two possible problem spaces are in 2D image classification (Fu, 1982) and in human-made scene classification (Han and Zhu, 2009). Each of these problems have been described by a formal categorizing hierarchy-based structure similar to a formal grammar. One final problem space is musical notation which already has a well-established grammar. Problems that deal with short segments of music are likely to perform the best due to the signal decay of recursive architectures.

8.4 Summary of Contributions

This thesis describes a novel machine learning device suitable for formal language string classification, the Neural Grammar Network (NGN). After some adjustment to training design and problem analysis, the NGN may become yet more competitive against its contemporaries in QSAR problem solvers. It has already done so for some specific cases, offering stability and comparable performance particularly in select classification tasks, and better average performance in many regression tasks. This work has included the development of a working example of NGN generating software, to which a user only has to submit a grammar and system parametres; implementation details and a mathematical formalization and finally the realization of the theoretical device in a practical application. Careful and thorough reporting on experimental findings along with analyses of results allows the continued future development of and exploration with the NGN.

References

Blair, R. M., Fang, H., Branham, W. S., Hass, B. S., Dial, S. L., Moland, C. L., Tong, W., Shi, L., Perkins, R., and Sheehan, D. M. (2000). The estrogen receptor relative binding affinities of 188 natural and xenochemicals: structural diversity of ligands. *Toxicol. Sci.*, 54(1):138–153.

Bohm, M., Sturzebecher, J., and Klebe, G. (1999). Three-dimensional quantitative structure-activity relationship analyses using comparative molecular field analysis and comparative molecular similarity indices analysis to elucidate selectivity differences of inhibitors binding to trypsin, thrombin and factor xa. *J. Med. Chem.*, 42:458–477.

Branham, W. S., Dial, S. L., Moland, C. L., Hass, B. S., Blair, R. M., Fang, H., Shi, L., Tong, W., Perkins, R. G., and Sheehan, D. M. (2002). Phytoestrogen and mycoestrogen bind to the rat uterine estrogen receptor. *Journal of Nutrition*, 132(4):658–664.

Bruce, C. L., Melville, J. L., Pickett, S. D., and Hirst, J. D. (2007). Comtemporary qsar classifiers compared. *Journal of Chemical Information and Modeling*, 47(1):219–227.

Ceroni, A., Costa, F., and Frasconi, P. (2007). Classification of small molecules by two-and three-dimensional decomposition kernels. *Bioinformatics*, 23(16):2038–2045.

Dalby, A., Nourse, J. G., Hounshell, W. D., Gushurst, A. K. I., Grier, D. L., Leland, B. A., and Laufer, J. (1992). Description of several chemical structure file formats used by computer programs developed at molecular design limited. *Journal of chemical information and computer sciences*, 32(3):244–255.

de A. Barreto, G., Araújo, A. F. R., and Kremer, S. C. (2003). A taxonomy for spatiotemporal connectionist networks revisited: the unsupervised case. *Neural Comput.*, 15(6):1255–1320.

Depriest, S. A., Mayer, D., Naylor, C. B., and Marshall, G. R. (1993). 3d-qsar of angiotensin-converting enzyme and thermolysin inhibitors - a comparison of comfa models based on deduced and experimentally determined active-site geometries. *J. Am. Chem. Soc.*, 115:5372–5384.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

Fang, H., Tong, W. D., Branham, W. S., Moland, C. L., Dial, S. L., Hong, H. X., Xie, Q., Perkins, R., Owens, W., and Sheehan, D. M. (2003). Study of 202 natural, synthetic and environmental chemicals for binding to the androgen receptor. *Chem Res Toxicol*, 16(10):1338–1358.

flex (2008). flex - the fast lexical analyzer. http://flex.sourceforge.net/.

Fountaine, F., Pastor, M., Zamora, I., and Sanz, F. (2005). Anchor-grind: Filling the gap between standard 3d qsar and the grid-independent descriptors. *J. Med. Chem.*, 48(7):2687–2694.

Frasconi, P., Gori, M., Kuechler, A., and Sperduti, A. (2001). From sequences to data structures: Theory and applications. In Kolen, J. and Kremer, S., editors, *A Field Guide to Dynamic Recurrent Networks*, pages 351–374. IEEE Press.

Fu, K.-S. (1982). *Syntactic Pattern Recognition and Applications*. Longman Higher Education.

GNU bison (2008). Bison - gnu parser generator. http://www.gnu.org/software/bison/.

Gohlke, H. and Klebe, G. (2002). Drugscore meets comfa: Adaptation of fields for molecular comparison (afmoc) or how to tailor knowledge-based pair-potentials to a particular protein. *J. Med. Chem.*, (45):4153–4170.

Guha, R., Howard, M. T., Hutchison, G. R., Murray-Rust, P., Rzepa, H., Steinbeck, C., Wegner, J. K., and Willighagen, E. (2006). The blue obelisk – interoperability in chemical informatics. *J. Chem. Inf. Model*, 46(3):991–998.

Hadley, R. F. and Hayward, M. B. (1997). Strong semantic systematicity from hebbian connectionist learning. *Minds Mach.*, 7(1):1–37.

Han, F. and Zhu, S.-C. (2009). Bottom-up/top-down image parsing with attribute grammar. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(1):59–73.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Prentice Hall.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kolen, J. and Kremer, S., editors, *A Field Guide to Dynamic Recurrent Networks*, pages 237–243. IEEE Press.

Hochreiter, S. and Schmidhuber, J. (1996). Lstm can solve hard long time lag problems. In Mozer, M., Jordan, M. I., and Petsche, T., editors, *NIPS*, pages 473–479. MIT Press.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366.

James, C. A. (2007). Opensmiles specification. http://www.opensmiles.org/spec/opensmiles.html.

Jorissen, R. N. and Gilson, M. K. (2005). Virtual screening of molecular databases using a support vector machine. *Journal of Chemical Information and Modeling*, 45(3):549–561.

Karray, F. O. and Silva, C. W. D. (2004). Chapter 4: Fundamentals of artificial neural networks. In *Soft Computing and Intelligent Systems Design: Theory, Tools and Applications*, pages 221 – 248. Addison Wesley.

Karthikeyan, M., Glen, R. C., and Bender, A. (2005). General melting point prediction based on a diverse compound data set and artificial neural networks. *J. Chem. Inf. Model.*, 45(3):581–590.

Kauffman, G. W. and Jurs, P. C. (2001). Qsar and k-nearest neighbor classification analysis of selective cyclooxygenase-2 inhibitors using topologically-based numerical descriptors. *J. Chem. Inf. Comput. Sci.*, 41(6):1553–1560.

Kemke, C. (2002). A constructive approach to parsing with neural networks - the hybrid connectionist parsing method. In *Advances in Artificial Intelligence*, volume 2338, pages 310–318. Springer Berlin / Heidelberg.

Kernighan, B. W., Ritchie, D., and Ritchie, D. M. (1988). *C Programming Language (2nd Edition)*. Prentice Hall PTR.

Kolen, J. F. and Kremer, S. C., editors (2001). *A Field Guide to Dynamical Recurrent Networks*. Wiley-IEEE Press.

Kremer, S. C. (2001). Spatiotemporal connectionist networks: A taxonomy and review. *Neural Comput.*, 13(2):249–306.

Lane, P. C. and Henderson, J. B. (2003). Towards effective parsing with neural networks: Inherent generalizations and bounded resource effects. *Applied Intelligence*, 19:83–99.

Lane, P. C. R. and Henderson, J. B. (1998). Simple synchrony networks: Learning to parse natural language with temporal synchrony variable binding. In *In Proceedings of the International Conference on Artificial Neural Networks*, pages 615–620.

Li, H., Yap, C. W., Ung, C. Y., Xue, Y., Cao, Z. W., and Chen, Y. Z. (2005). Effect of selection of molecular descriptors on the prediction of blood-brain barrier penetrating and nonpenetrating agents by statistical learning methods. *J. Chem. Model.*, 45(5):1376–1384.

Miikkulainen, R. (1996). Subsymbolic case-role analysis of sentences with embedded clauses. *Cognitive Science*, 20:47–73.

Mohr, J. A., Jain, B. J., and Obermayer, K. (2008). Molecule kernels: A descriptor- and alignment-free quantitative structure-activity relationship approach. *Journal of Chemical Information and Modeling*, 48(9):1868–1881.

NCTR (2007). Nctr center for toxicoinformatics - edkb home page. http://www.fda.gov/nctr/science/centers/toxicoinformatics/edkb/index.htm.

OpenBabel (2008). The open babel package, version 2.1.1. www.openbabel.org.

Palmer-Brown, D., Tepper, J. A., and Powell, H. M. (2002). Connectionist natural language parsing. *Trends in Cognitive Sciences*, 6(10):437–442.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46:77–105.

Ralaivola, L., Swamidass, S. J., Saigo, H., and Baldi, P. (2005). Graph kernels for chemical informatics. *Neural Netw.*, 18(8):1093–1110.

Rumberlhart, D., Hinton, G., and Williams, R. (1986). Chapter 9: Learning internal representation by error propagation. In McClelland, J. L., Rumelhart, D., and the P.D.P. Group (Eds.), editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. MIT Press, Cambridge, MA.

Sejnowski, T. J. C. R. (1988). Nettalk: A parallel network that learns to read aloud.

Sharkey, N. E., Sharkey, A. J. C., and Jackson, S. A. (1994). Opening the black box of connectionist nets: Some lessons from cognitive science. *Computer Standards & Interfaces*, 16(3):279 – 293.

Shastri, L. and Ajjanagadde, V. (1993). From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16:417–494.

Shi, L. M., Fang, H., Tong, W., Wu, J., Perkins, R., Blair, R. M., Branham, W. S., Dial, S. L., Moland, C. L., and Sheehan, D. M. (2001). Qsar models using a large diverse set of estrogens. *ACS Publications*, 41(1).

Shiu, H. K., Chan, E., and Wan, L. (1996). Confluent preorder parser as a finite state automata.

Stein, S. E., Heller, S. R., and Tchekhovskoi, D. V. (2006). *The IUPAC Chemical Identifier Technical Manual*. Gaithersburg, Maryland, USA. http://old.iupac.org/inchi/download/index.html.

Sutherland, J. J., O'Brien, L. A., and Weaver, D. F. (2003). Spline fitting with a genetic algorithm: A method for developing classifiction structure-activity relationships. *J. Chem. Inf. Comput. Sci.*, 43(6):1906–1915.

Sutherland, J. J., O'Brien, L. A., and Weaver, D. F. (2004). A comparison of methods for modeling quantitative structure-activity relationships. *J. Med. Chem.*, 47(22):5541–5554.

Tepper, J. A., Powell, H. M., and Palmer-Brown, D. (2002). A corpus-based connectionist architecture for large-scale natural language parsing. *Connection Science*, 14(2):93–114.

Tong, W., Xie, Q., Hong, H., Shi, L., Fang, H., and Perkins, R. (2004). Assessment of prediction confidence and domain extrapolation of two structure-activity relationship models for predicting estrogen receptor activity. *Environmental Health Perspectives*, 112(12).

van Rossum, G. (2008). Python programming language – official website. http://www.python.org.

Vighi, M., Migliorati, S., and Monti, G. S. (2009). Toxicity on the luminescent bacterium *Vibrio fischeri* (beijerinck). i: Qsar equation for narcotics and polar narcotics. *Ecotoxicity and Environmental Safety*, 72:154–161.

Walsh, I., Vullo, A., and Pollastri, G. (2009). Recursive neural networks for undirected graphs for learning molecular endpoints (submitted).

Werbos, P. J. (1994). *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*. Wiley-Interscience, New York, NY, USA.

Wheeler, D., Barrett, T., Benson, D., Bryant, S., Canese, K., Chetvernin, V., Church, D., Dicuccio, M., Edgar, R., Federhen, S., Feolo, M., Geer, L., Helmberg, W., Kapustin, Y.,

Khovayko, O., Landsman, D., Lipman, D., Madden, T., Maglott, D., Miller, V., Ostell, J., Pruitt, K., Schuler, G., Shumway, M., Sequeira, E., Sherry, S., Sirotkin, K., Souvorov, A., Starchenko, G., Tatusov, R., Tatusova, T., Wagner, L., and Yaschenko, E. (2008). Database resources of the national center for biotechnology information. *Nucleic Acids Res.*, 36. http://pubchem.ncbi.nlm.nih.gov.