NSL: Neural Network Specification Language

Tom Borchert
ECEN 5523
Semester Project

December 9, 2005

NSL is a specification language for neural networks. It allows the user to specify a network based on its topology and its behavior. The output is a C file that implements the specified model.

1 Language Definition

In NSL a network is described as a series of layers. Each layer has four properties:

- **NODES**: the number of nodes in the layer.
- **ACTIVATION UPDATE**: the function that describes how node activation values are to be computed.
- **WEIGHT UPDATE**: the function that describes how connection weights are to be updated.
- **WEIGHT INIT**: specifies the initial values of the connection weights.
- Layers are optionally specified to be input or output layers.

So, the description of a single layer looks like this:

```
[INPUT / OUTPUT] LAYER layername
    NODES: expression;
    ACTIVATION_UPDATE: expression;
    WEIGHT_UPDATE: expression;
    WEIGHT_INIT: expression;
ENDLAYER
```
Global values for all of these properties except \texttt{ACTIVATION\_UPDATE} may be specified in the source file before any layer constructs. If an individual layer description is missing any property, it will use the global value for that property. If the global value is not found, an error occurs.

The connections between layers are not specified explicitly, but are implicit in the \texttt{ACTIVATION\_UPDATE} function. For example, if we have

\begin{verbatim}
LAYER B
  ACTIVATION\_UPDATE: SUM(A)
ENDLAYER
\end{verbatim}

then there must be connections between every node in layer \texttt{A} and every node in layer \texttt{B}.

\section*{1.1 Layer Properties}

The following properties specify the behavior of a single layer in the network. Note that all of these properties except \texttt{ACTIVATION\_UPDATE} can be omitted if they are specified globally.

\subsection*{1.1.1 \texttt{ACTIVATION\_UPDATE}}

When updating the activation value for the nodes in a layer, the \texttt{ACTIVATION\_UPDATE} expression is applied to every node individually. The predefined function \texttt{SUM(A)} computes sum of the nodes in layer \texttt{A} weighted by the connection weights between those nodes and the node currently being computed. The predefined variable \texttt{PREV} references the layer that occurs immediately before this one in the source file.

When another layer is referenced in the \texttt{ACTIVATION\_UPDATE} expression, a set of weights is created from that layer to the one being described.

\subsection*{1.1.2 \texttt{WEIGHT\_UPDATE}}

The \texttt{WEIGHT\_UPDATE} expression for a layer is applied to every connection leading into that layer. Every connection has a source node in the previous layer, and a destination node in the current layer.

The following predefined expressions can be used in a \texttt{WEIGHT\_UPDATE} expression:

\begin{itemize}
  \item \texttt{IN\_VAL}: the current activation value of the connection’s source node.
\end{itemize}
• OUT\_VAL: the current activation value of the connection’s destination node.

• ERROR: the error assigned to this node. For output nodes, this is the difference between the node activation value and the desired output value. For other nodes, this is the amount of output error the node is responsible for, based on the connection weights between this node and the output.

1.1.3 WEIGHT\_INIT

Every connection leading into a layer has its weight initialized using the layer’s WEIGHT\_INIT expression. The predefined function RANDOM(min, max) can be used to specify a random number in a given range.

1.2 Global Declarations

The following properties apply to the entire network.

1.2.1 STOP\_WHEN

This tells the model when it should halt the learning algorithm, and it must be included in the global declarations. Generally learning halts when total error is below some threshold, or when training has gone on for too long. The following predefined variables can be used in the STOP\_WHEN expression:

• SQ\_ERROR: the total squared error when the output of the network is compared with the desired outputs specified in the training data.

• ABS\_ERROR: the total absolute error of the model compared to the training data.

• EPOCH: the number times the learning algorithm has looped through the entire training set.

1.2.2 Global Variables

Using the syntax:

variable name: expression;

The user can specify a variable that can then be referenced in other expressions. This expression will be copied verbatim any time the variable is used in an expression.
1.2.3 External Functions

The user may declare external functions, specified as C functions in external files, that can then be used in expressions.

\[ \text{function: } "\text{filename}"; \]

makes \textit{function} available for use in any expression, and will cause the specified file to be included in the output C code.

1.3 Model Parameters

The first line of a specification must be a list of the model parameters. For example:

\textsc{PARAMS} error\_threshold \textsc{max}\_epoch;

These are values that will be specified on the command line when running the model, and they can be used as variables in any expression. The parameters specified in the above example could be used in the \textsc{STOP\_WHEN} expression as follows:

\textsc{STOP\_WHEN}: SQ\_ERROR < error\_threshold OR EPOCH > max\_epoch;

1.4 Expressions

The expressions used to specify the layer properties and global declarations can be a combination of mathematical formulas and function calls. The following operators are allowed in these expressions: \(+\), \(-\), \(*\), \(/\), \(<\), \(<=\), \(>\), \(>=\), and boolean \textsc{AND} and \textsc{OR}. Standard precedence rules are used when evaluating the expressions.

Using a variable that is not a layer name, a predefined entity, a model parameter, or declared in the global declarations section, is an error. Using a function that is not one of the predefined functions and is not declared in the global declarations section is also an error.

2 Structural Analysis

2.1 Abstract Syntax Tree

A network contains a list of parameters, a list of global declarations, and a list of layers.
2.2 Layers

Each layer is made up of a layer name and a list of declarations. These declarations usually include the layer properties, but can also be declarations of variables or external functions.

In several places NSL needs to get a property specifically from the input or output layer. For this reason these layers get their own symbols in the AST.

2.3 Expressions

Expressions are copied verbatim into the output C file, but are parsed into the AST anyways. This is for two reasons. The first is that NSL needs to know when a layer is referenced in an \texttt{ACTIVATION UPDATE} expression so that it can generate the appropriate weights. The second is that we want to be able to perform name analysis to ensure that all variables referenced in the expressions are properly declared.

Each different type of expression is specified as its own rule in the AST. This was done to keep things simple for the limited number of math and logic expressions that are available. Also, parentheses in expressions are denoted by square brackets instead of normal parentheses to make it easier for the AST to distinguish between function calls and parentheses.

3 Semantic Analysis

The four layer properties, \texttt{NODES}, \texttt{ACTIVATION UPDATE}, \texttt{WEIGHT INIT}, and \texttt{WEIGHT UPDATE}, are PTGNode attributes of every layer, and also of the entire network. If a property is not specified in the layer, then the property is retrieved from the network.

Layers, model parameters, variables, and external functions are treated as identifiers using the \texttt{IdDefScope} role. This allows the VarUse expression to make sure that no undeclared functions or variables are referenced by using the \texttt{ChkIdUse} and \texttt{IdUseEnv} roles. It also allows the VarUse expression to get the properties of a layer, if the identifier it contains is a layer name.

Every variable and number in an expression is assumed to be of type \texttt{double}, so no type analysis is required.

The files referenced in an external function declaration are included in the output using the \texttt{#include} directive. To ensure that a file is only included once, the FileName symbol uses the \texttt{IdDefScope} role. A \texttt{IsIncluded} property is set the
first time the file is included, and then checked before outputting the \texttt{\#include}
statement the next time.

### 3.1 Layers

Each layer defines a new scope, using the \texttt{RangeScope} role.

The \texttt{IsLayer} property is used to distinguish between identifiers that are layers
and identifiers that are functions or variables. There is no way to tell during
structural analysis whether an identifier in an expression is a layer or a variable,
so this property is set during semantic analysis.

There are times when the code generated to perform operations on layers needs
to be executed in a certain order. This is necessary when updating the node
activation values, when updating the connection weights, and when saving and
loading the layers from the model file. A different PTGNode chain is used for
each of these circumstances, and the layers add their code either at the beginning
or the end of the current chain value depending on what order the execution
needs to occur in. When the order of execution does not matter, for instance
when allocating space for the node values, \texttt{CONSTITUENTS} is used instead of
a chain.

### 3.2 Connection Weights

All output dealing with layer connections is generated in the VarUse expression
rule. This rule is the only place that there is easy access to both the layer
containing the expression and the layer being referenced by the expression. Also,
many of these references to connection weights must be output in a specific
order, either in textual order or in reverse textual order. A lot of different
chains are used, one for each time that connection weights must be included
in the output C code. These chains are PTG nodes that are appended to by
VarUse expressions that reference another layer.

The VarUse expression already contains the name of the referenced layer, but
in many cases it also needs to know the number of nodes in that layer. The \texttt{NodeCount}
property is set by each layer and then used by the VarUse expression
when it references that layer.

### 4 Constraints

Every layer ensures that it contains a \texttt{ACTIVATE,UPDATE} specification. It also
ensures that the other three layer properties are either contained in that layer,
or contained above the layer in the specification.
The network ensures that it contains a `STOP WHEN` specification.

Every variable and function reference in an expression must be declared somewhere in the source file.

Any violation of these constraints causes the translator to fail and display an error.

5 Example Source File

```plaintext
PARAMS learn_rate hidden err_threshold;

WEIGHT_INIT: RANDOM(0, 2);
WEIGHT_UPDATE: learn_rate * ERROR * IN_VAL * OUT_VAL * [1 - OUT_VAL];
STOP WHEN: SQ_ERROR < err_threshold;

INPUT LAYER A
   NODES: 50;
ENDLAYER

LAYER B
   NODES: hidden;
      ACTIVATION UPDATE: sigmoid(SUM(A));
ENDLAYER

OUTPUT LAYER C
   NODES: 50;
      ACTIVATION UPDATE: sigmoid(SUM(B) + SUM(A));
ENDLAYER
```