

# Neural Network for recognizing message context

Scott Auge

Amduus Information Works, Inc.

[sauge@amduus.com](mailto:sauge@amduus.com)

Version 1.3

## Theory of operation:

(If you want to treat the neural net as a black box, go to the bottom of this page to see the procedures to call. Remember – train first, use after training.)

There are many messages written in human language that may need to be classified. An example of this might be email messages containing unsolicited commercial messages (i.e., Spam.) Other less obvious examples may be emails written from customers of a large corporation: is the email meant for finance? The web administrator? The sales group? The service group?

When we examine a message, we see that certain words are used in combination to express the thoughts of the writer. A common approach used to recognize the meaning of the message is to use word matching to determine if the message is what it is. *What is missing from this approach is the context of the message.* If one sets up a rule saying “if the word ‘sale’ is in the message then the message is for sales,” it can be wrong simply because the word sale might be used in the context of the sender wishing to tell about a sale, or it could be asking for a sale.

These kinds of word existence rules soon become complicated and really do not sort out messages very well. This is not a problem that is easily solved with Boolean/predicate calculus logic.

The neural network operates on collections of words, versus a set of rules on words. Certain types of messages will always have a collection of words that need to be used to express the thought and intent of the writer. But writers generally do not use the same words in the same order to express the idea. Messages are very fuzzy in their use of different expressions to result in the same output.

How the NN (Neural Net) represents this is very different from the Predicate calculus method of determining the classification of a message. The NN is trained with inputs from messages that have been classified as the type of classification the NN is to learn. This is called training data.

This data and algorithms associated with the NN will create an NN that should recognize this type of data in the future.

Another aspect of the neural network is that it does not give a definitive answer that the message is what it thinks it is. But it will give a number that when large represents it thinks it likely to be the classification it has learned, and when low represents it thinks it likely not to be a classification it has learned.

One will need to create a threshold value where a message is considered a member of the classification, and when it is not. If the message is a member of the classification, it should be turned into training data and feed back into the NN.

Here is a simple explanation of how the NN works in this implementation.

Each word is sent as an input to the neural network. Each word of training data creates a neuron. Associated with this neuron is a numeric weight that describes the frequency of the word used in the training data. Then the collection of words is examined at whole to determine if a high frequency of words used in the message have been presented in the training data.

here is an simple example:

Here is three sets of training data:

```
a f h u i  
k u f o p  
r t u k l
```

This could be an abstracted view of spam messages.

After training, the NN will have neurons that associate the frequency of the words. In the above training data, it would end up with this:

```
Na = 1  
Nf = 2  
Nh = 1  
Nu = 2  
Ni = 1  
Nk = 2  
No = 1  
Np = 1  
Nr = 1  
Nt = 1  
Nl = 1
```

Now a message is sent into the NN to determine if it is of the classification found in the training data. If enough training data is presented, it should be able to determine associatively if the message is part of that set.

f u k o w

is the message to be classified...

We see that the elements (words) f u k o are common in the training set. A message containing these words are very likely to be of the training set. And the NN would return an output of a large number.

The message

q j h t x

is not part of the training set, and so the NN should return a low number.

This approach uses a feed-forward neural network based on the perceptron with a Log-Sigmoid transfer rule. The  $\sum$  is slightly modified for those vectors that are not present. Under that circumstance, the value is subtracted by the default weight of a neuron (best represents a missing neuron I suppose.)

One aspect of this approach that differs from most neural network implantations is the automatic creation of neurons in the first layer. Once created, the network will use these neurons for additional learning about messages presented to it.

### **Neuron representation:**

The neurons are represented in a table called Neuron. It is composed of the following fields:

*NeuronID* – a unique identifier for the neuron record

*Layer* – for multi-level networks, this field represents which layer the neuron is a member of

*Weight* – Weight of the neuron learning

*Bias* – Bias of the neuron response

*Data* – Data that the neuron knows about

*Name* – the table may contain multiple NN's. This is to differentiate between them.

### **Teaching the NN:**

An input of words are presented to the net. It performs the following:

1. Pull a word out of the message.
2. Does the neuron for this word exist? If not, create it. Else find it.
3. Use the learning rule to adjust the weight of the neuron.
4. Go to step 1.

### **The learning rule:**

$$Nw = Nw + t$$

where  $t = 1 + n$  where  $0 < n < 1$

The larger n, the more hyper the learning, the lower, the slower the learning.

### **Actuating Bias:**

Some words are very common to both the message to be classified and the message outside that classification.

The bias can be trained with messages outside the classification the NN is to recognize, but should not create neurons for inputs not common to both message types.

The Bias should be a negative number in these conditions.

The Rule is

$$\text{Bias} = \begin{cases} \text{if Bias} > 0 \rightarrow \text{abs}(\text{Bias}) * (1 + t) * -1 \\ \text{else Bias} = 1 \end{cases}$$

where  $t = 1 + n$  where  $0 < n < 1$

### **The transfer rule:**

The Log-Sigmoid rule is used in this NN. It can take plus to minus infinity and squash it to 1 to 0. The rule is defined as:

$$a = \frac{1}{1 + e^{-n}}$$

Use of the function would be:

$a = f(n)$  where  $n = w + b$

and

$w = \text{Weight}$

$b = \text{Bias}$

### **Tuning the Neural Network:**

In the default.i file there are tunable parameters for the network.

```
&GLOBAL-DEFI NE DFLTWEI GHT      0. 5
&GLOBAL-DEFI NE DFLTBI AS        0. 0
&GLOBAL-DEFI NE DFLT LAYER       " 1"
&GLOBAL-DEFI NE DFLTTRAI N       1. 2
&GLOBAL-DEFI NE DFLTSETBI AS     -0. 4
&GLOBAL-DEFI NE DFLTTRAI NBI AS  0. 25
```

DFLTWEIGHT

DFLTTRAIN

For large messages, these should be set low. There will be plenty of material for the NN to digest. Remember DFTLTRAIN > 1 must always be true.

If the messages are short, not containing many words, one will want to keep the weights about where they are at.

*The following routines make the NN a black box for programmers who just want to use a NN and not interested in understanding the theory.*

### **Calling the NN:**

RUN UseNN.p (INPUT Message, INPUT Name, OUTPUT IsClassification, OUTPUT TrxFunction)

CHARACTER Message – Message to check if it is part of the classification NN learned

CHARACTER Name – Name of the NN

LOGICAL IsClassification – YES/NO of it the message is part of the classification

DECIMAL TrxFUNCTION – Neural Net’s real answer

**Training the NN:**

RUN TrainNN.p (INPUT Message, INPUT Name)

CHARACTER Message – Message that is part of the classification for the NN to learn  
CHARACTER Name – Name of the NN

**Setting Bias:**

RUN SetBias (INPUT Message, INPUT Name)

CHARACTER Message – Message that is part of the classification for the NN to learn  
CHARACTER Name – Name of the NN

**Clearing a Neural Net:**

RUN ForgetNN (INPUT Name)

CHARACTER NAME – Name of the neural net to blow away