

MPSMD3ART Workshop 1, Introduction

Dr Ian Hocking

Contents

1	Introduction to Tlearn	1
2	Objectives	1
3	About this document	1
4	Workshop	3
5	Versions	21

1 Introduction to Tlearn

<i>Data files required</i>	Download here
<i>Booklet Version</i>	1.0
<i>Format</i>	Standard PDF

2 Objectives

You will use a simple network design to learn about Tlearn, basic modelling principles, parameters, and the limitations of certain model designs.

3 About this document

This document is available in different formats for students who may have accessibility requirements. See [Versions](#). The system is still being piloted and I'd be interested in your [feedback](#).

3.1 Tasks and Your Lab Journal

Use this booklet in conjunction with your own *Lab Journal*, where you will record your workings, thoughts, and other comments related to the exercises. Your Lab Journal can take any form, but a Word document might be best; you can copy and paste output from Tlearn alongside your notes.

(If you're looking at a non-standard, accessible version of this document, some of the formatting below will be simplified.)

- When I ask you to complete a task, like calculate a mean, it will be formatted like this.

- This is what a Lab Journal reminder looks like. I'll use these when asking you to make a note.

3.2 Other Aspects of this Booklet

- This formatting will be used to highlight something important.

Answer

Here I'll provide answers to questions. Note that this version of the document won't be available until after your workshop.

3.3 Downloading Tlearn

You can download Tlearn for Windows, Mac and Unix here: [Innate](#).

You might also want to download the [Tlearn manual](#).

Remember that you're just setting out with Tlearn, so this booklet might take you some time to get through. However, it's important that you work through it methodically and try to understand as much as possible as you go along.

3.4 Answers

Sometimes, when there is an instructor version of an exercise booklet, you'll be provided with a second version of this document, containing answers, a few days after your seminar.

4 Workshop

Tlearn is a simple connectionist network program developed by Kim Plunkett and Jeffrey L. Elman. It's a bit long in the tooth, but it is much more straightforward to use than most connectionist software, which makes it suitable as a teaching tool. That said, please remember:

1. Save your work regularly
2. Keep backups of the files you're working on.

Tlearn hasn't been actively developed since 1999, and it does have some bugs.

4.1 Level of Computing Knowledge Assumed

Not much! You should know about *files* and *file types*. Don't get frustrated; if something isn't working the way you expect, ask someone!

Generally speaking, Tlearn reads and writes *text files*. These are very simple files that contain (for the most part) only numbers and letters, no formatting. When you want to edit text files, I'd advise you to use a *text editor* like Windows Notepad. Microsoft Word won't work as well with text files as it tends to save formatting information in the file that confuses Tlearn.

4.2 How Tlearn Works

In a moment, we'll run through an interactive example of Tlearn. But, right now, I want to give you a quick overview of how Tlearn works.

The Tlearn program works like a dashboard where you manage several files simultaneously (i.e. a single project contains several files). This contrasts with a program like Microsoft Word, in which you tend to focus on one file at a time.

For each project (i.e. connectionist model), Tlearn will use several files.

Let's say you want you're working on a model of past tense acquisition. You should think of a meaningful name for the Tlearn project file, like *tense*. When you ask Tlearn to create a new project called tense, the following files are created:

- *tense*

- *tense.cf*
- *tense.data*
- *tense.teach*

Although each has a different file extension (e.g. *.data*), all files are plain text. Let's look at each of these in turn.

4.3 tense

This file is created automatically when you ask Tlearn to create a new project.

4.4 tense.cf

This is a *configuration file*. It describes how the connectionist model looks and behaves. Tlearn expects the text in this file to follow a rigid format, which you'll learn about below.

4.5 tense.data

This contains the input data—usually numeric—that Tlearn will present to the network when it learns. In the case of a network learning the past tense, the data might well be the present tense forms of a verb.

4.6 tense.teach

This file contains the output patterns that you wish to associate with each input pattern. These output patterns are the desired outputs. For instance, if an input is the present tense verb to go, the desired output might be went.

Once Tlearn has done some work by training a network, a snapshot of the 'weights' are saved. These weights represent the strength of the connections between 'nodes'. The file is named using the project title, the number of sweeps (iterations through a training set), and the extension 'wts', like this: *tense.1000.wts*.

If you've asked Tlearn to record errors, you'll get a file with the extension 'err': *tense.err*.

4.7 The AND Function

The logical AND function is an important part of how a computer works. Along with functions like OR, and XOR, it is a building block for increasingly complex

forms of computation. Its simplicity makes it a good place to start, even though its relevance might be challenging to appreciate!

For this exercise, you'll need to use files specified [here](#).

The logical AND function can be described in plain English as 'when two inputs are active, make an output active'. This is a very abstract way of describing it. Another AND function might be 'When I am at home and the temperature in my house drops below 18 degrees celsius, turn on the central heating'.

This is what's happening in Table 2. When input 1 is true (i.e. switched on) and input 2 is true (switched on), then make the output true; otherwise do nothing.

Table 2: The Logical AND Function

Input 1	Input 2	Output
0	0	0
1	0	0
0	1	0
1	1	1

In the above table, you can see that we've included all possible combinations of two inputs. The output column contains zeroes apart from the final line, where the inputs are '1' and '1'. So this function only 'returns' 1 when both inputs are '1': that is the logical AND function.

Let's begin.

- (1) From the 'Network' menu, use 'Open project' to open the AND project file (that's the one without the filename extension). See Figure 1.

By the way, Tlearn has a bug in the way it handles filenames. It may crash when opening a file whose pathname has spaces or refers to a network location. So opening a file on my Windows desktop causes a crash. My suggested solution is to save files to a thumbdrive.

Once the project file has been opened, your screen should look something like Figure 2.

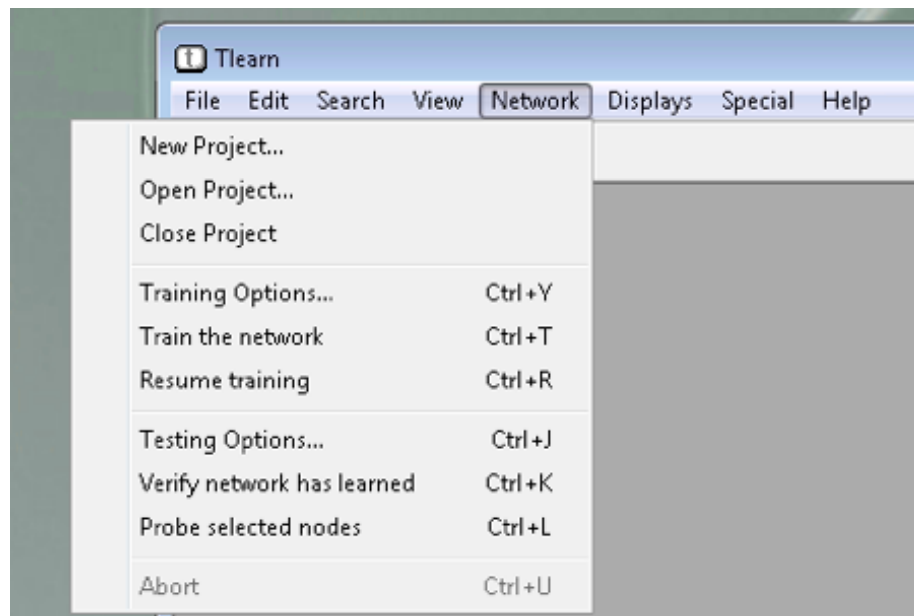


Figure 1: Tlearn Open Project

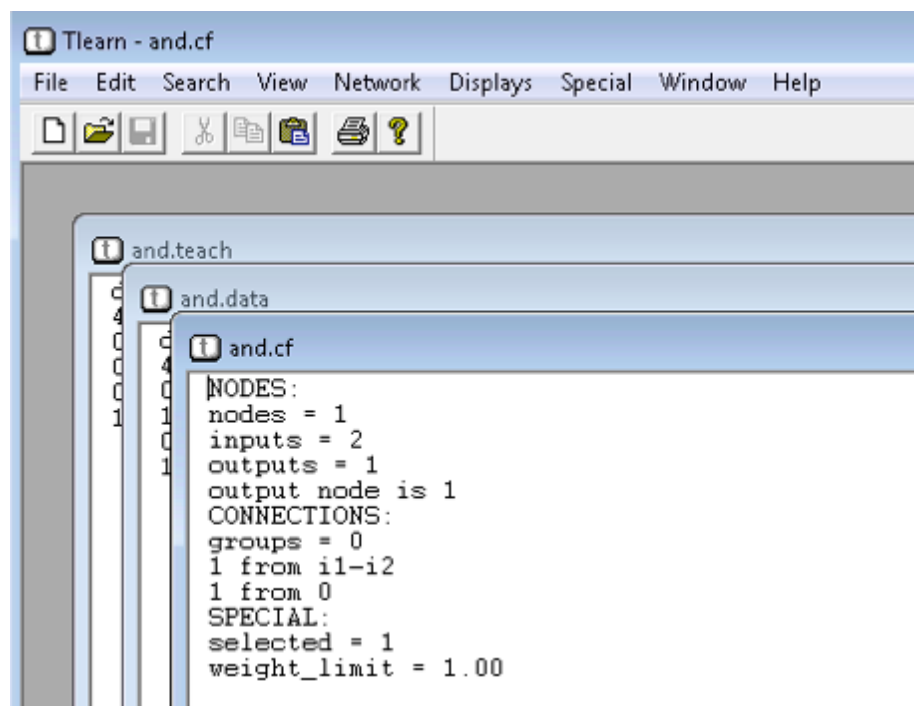


Figure 2: Tlearn Files Open

- (2) Have a look at the network. Using the ‘Displays’ menu, open the network architecture, weights and activation displays.

These windows aren’t terribly informative right now because no learning has taken place. Let’s train the network to get some performance data.

- (3) Using the ‘Display > Training Options...’ menu, make sure that your settings look like mine in Figure 3.
 - Ignore the “and” run 2 of 2 at the top of the box; yours might be slightly different.

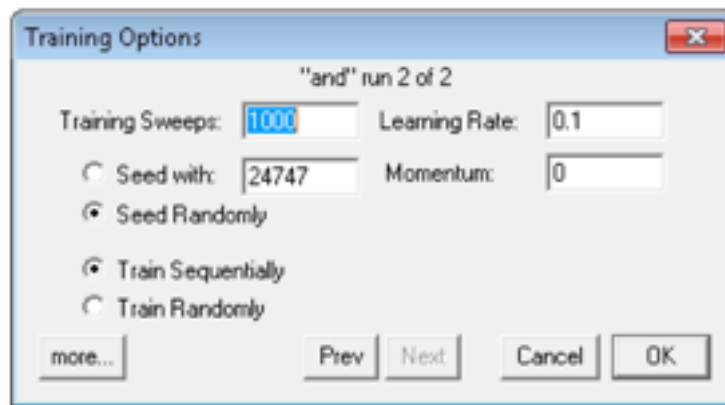


Figure 3: Tlearn Training Options

Let’s talk briefly about these parameters.

4.8 Training sweeps

In the logical AND function, one sweep is one presentation of an input and an output (e.g. input “0, 0” to output “0”). When four sweeps have been presented (i.e. the whole training set), we call this one epoch.

4.9 Learning rate

This tells the network how much it should update its weighted connections when they need changing during the learning process. A lower learning rate makes

for a slow but fairly effective learning network; a higher rate makes a network that tries to learn faster, but sometimes less effectively.

4.10 Seed with/Seed randomly

If you use a ‘seed’, the weighted connections between nodes will be determined by the seed; basically, this means that networks using the same seed will have the same weighted connections when they start out. That’s useful if you want to compare two identical networks. Typically, however, we want networks to have random weighted connections when they start out.

4.11 Train Sequentially/Train randomly

The patterns in the .data file can be presented in their original order or in a random order.

We can see what our network looks like in Figure ??.

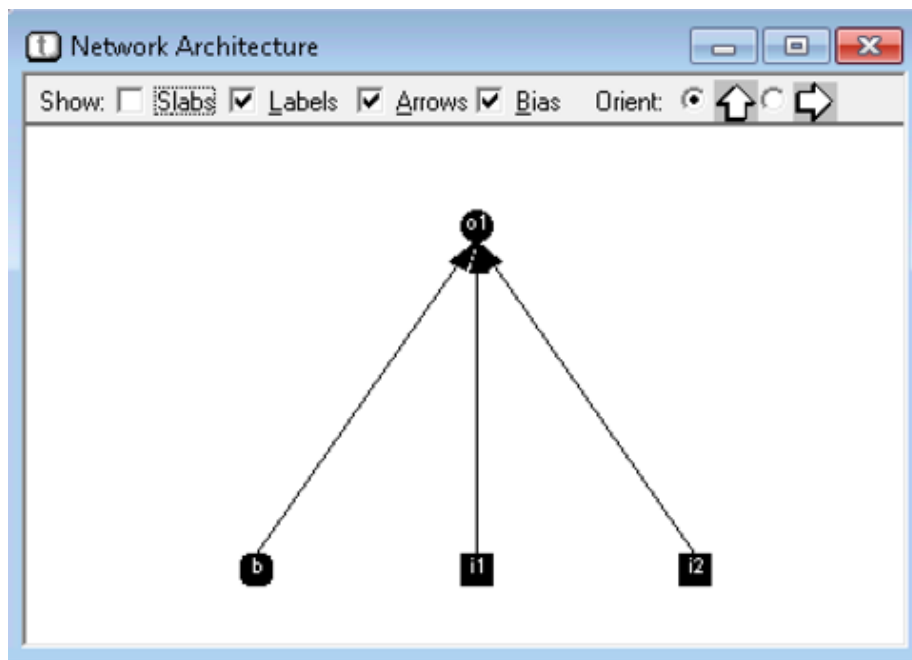


Figure 4: Tlearn Network Architecture

This tells us a few things.

4.12 o1

This is the output node, whose numerical value will vary depending on the input it receives from input nodes.

4.13 i1 and i2

These are the two input nodes. Values are presented to these two input nodes.

4.14 The arrows

This show the direction of activation or ‘flow’ between units.

4.15 b

This is called the ‘bias’. We don’t need to worry too much about it right now. It’s job, basically, is to help the output node vary its sensitivity (i.e. it’s activation threshold).

Why does the network have two inputs (ignoring the bias) and one output? That’s because we’re trying to model the AND function, which takes two inputs and gives one output (e.g. inputs of 1 and 1 to give an output of 1).

At the top of this window are several options that can change the physical appearance of the network.

This network architecture is not accidental. It looks like this because it was specified in the AND.cf file.

Let’s have a look at this file now.

(4) Select Window > and.cf

In this window, you should see something like this.

```
NODES:
nodes = 1
inputs = 2
outputs = 1
output node is 1
CONNECTIONS:
groups = 0
```

```
1 from i1-i2
1 from 0
SPECIAL:
selected = 1
weight_limit = 1.00
```

Before I explain how this works in detail, note that Tlearn is very picky about how this file is laid out. By picky I mean that Tlearn will get upset and crash if the file doesn't fit its expectations. Capitalisation, spacing, and punctuation are all very important. It also wants the file to be saved as in a pure text format. The best way of ensuring the format is pure text is to use either the built-in text editor within Tlearn (that's what you're using when you select the and.cf window), or the Windows program Notepad. You can find Notepad from the Start Menu. Later, when you're editing a .cf file to change the properties of a network, you might find that Tlearn crashes. If this is case, check that the layout of the file conforms to that of the .cf above, and check that it's pure text file (opening it in Notepad and saving should do the trick; never use Microsoft Word for this).

OK, so there are three sections to the and.cf file. NODES, CONNECTIONS, and SPECIAL. Let's consider each in turn. First, I'll explain what each bit does in general, and then I'll show how this relates to our AND network specifically.

4.16 NODES

This section indicates the number of nodes, the number of inputs, how many outputs, and which of the nodes should be considered as outputs. This will become clearer as we work through the example.

4.17 CONNECTIONS

This tells Tlearn how the bits of the network are connected. Though very simple in our AND example, network connections can get very complicated.

4.18 SPECIAL

Here we can tell Tlearn some extra things. One is the 'selected node', which just means the node for which we're interested in observing the values.

We'll now look once more at the .cf file. In Table 3, I'll explain what each part of the file means for us doing the AND function.

Table 3: Understanding the file ‘and.cf’

Instruction	Explanation
NODES:	This section contains info about the nodes
nodes = 1	There will be one computational node (i.e. a node that receives input)
inputs = 2	Create two input units
outputs = 1	There will be one output node
output node is 1	The output node is called node 1
CONNECTIONS:	This section contains info about how the network is wired up
groups = 0	Don’t create any groups of nodes
1 from i1-i2	Connect node one (which is the output node) to inputs i1 and i2
1 from 0	Connect node zero (which is the bias node) to node one (which is the output node)
SPECIAL:	This section contains info about the nodes
selected = 1	This designates node one (the output node) as the one we’re interested in testing for later
weight_limit = 1.00	The weighted connections in the network (between the input units and output node) will not be allowed to exceed 1.

OK, let’s train the network.

(5) Select Network > Train the Network

- Once you’ve done this, you should see Figure 5, which tells you that the network has done some learning.

This means that the logical AND input patterns, and their associated outputs, have been presented to the network 1000 times. Moreover, each time a pattern has been presented (in one sweep), the weighted connections have been updated.

(6) Find out how well the network has learned by selecting Window > Error Display

We can see how my network is doing by looking at Figure 6.

Unhelpfully, the Y axis is not labelled. What is it?

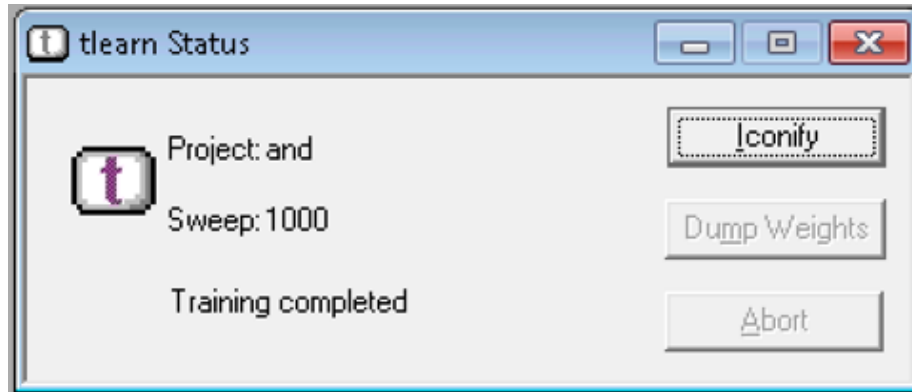


Figure 5: Tlearn Status

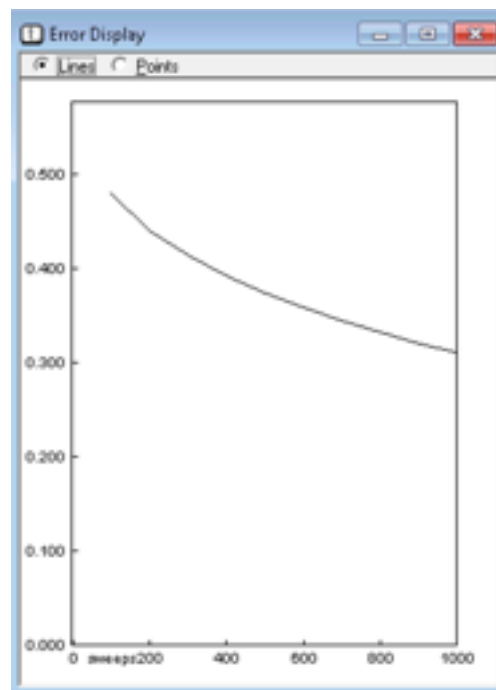


Figure 6: Tlearn Error Display

- (7) Find out by going to Network > Training Options, and clicking the ‘More’ box in the bottom left of the window. See Figure 7.

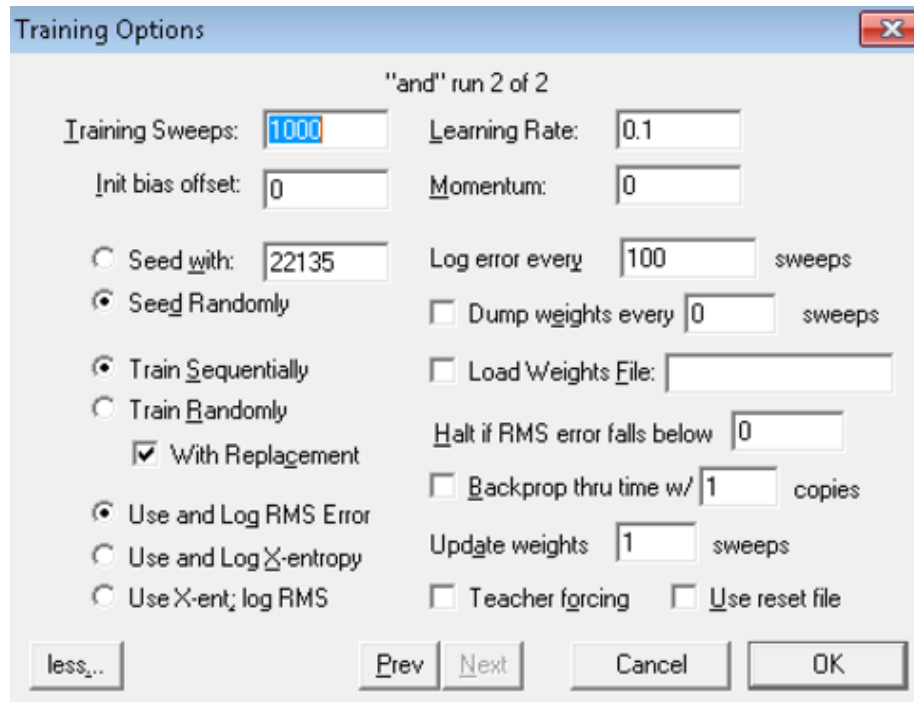


Figure 7: Tlearn Full Training Options

You should see ‘Use and Log RMS Error’ is checked. The Y-axis on the error graph, then, is the Root Mean Squared error. My network is showing an error of 0.3. What does this mean, and why are we using RMS?

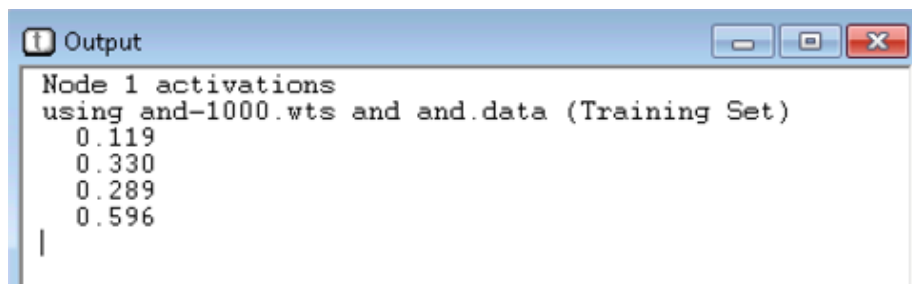
To recap, the logical AND function has four input patterns with associated outputs.

- (8) Get a snapshot of current output node performance by selecting Network > Probe Selected Nodes

For my network, the following window pops up (Figure 8). Your numbers will look slightly different.

The ‘Probe Selected Nodes’ tells us what’s happening with the output node

(node one) because we told Tlearn under the SPECIAL section of AND.cf to use node one as the ‘selected node’. Let’s look at this output in detail before illustrating how this relates to the error graph above.



```

Output
Node 1 activations
using and-1000.wts and and.data (Training Set)
0.119
0.330
0.289
0.596
|

```

Figure 8: Tlearn Output

Table 4 tells you what this file means.

Table 4: Understanding the output

Line	Explanation
Node 1 activations	The follow information is about node one (the output node)
using and-1000.wts and and.data (Training Set)	We’ll use the weights as they stand at 1000 sweeps (saved in ‘and-1000.wts’) and be presenting the patterns contained in the ‘and.data’ file
0.119	Node one (output node) activation when presented with inputs 0 and 0
0.330	Node one (output node) activation when presented with inputs 1 and 0
0.289	Node one (output node) activation when presented with inputs 0 and 1
0.596	Node one (output node) activation when presented with inputs 1 and 1

You can see from these numbers, and from the error graph itself, that the network is doing OK - not great, but OK. We really want the output for the first three patterns to be 0, and the output for the fourth pattern to 1. The network is heading in the right direction.

Bearing in mind the network has only had a thousand sweeps of learning, let’s use the above figures to work out how the network is doing in terms of an overall error statistic called the Root Mean Squared (RMS) Error. As you can see in Table 5, calculating RMS is straightforward.

Table 5: Understanding RMS (Root Mean Squared Error)

Input 1	Input 2	Desired Output	Actual Output	1. Error (Desired- Actual)	2. Error Squared (Error x Error)	3. Square Root
0	0	0	0.119	-0.119	0.014161	0.119
1	0	0	0.330	-0.33	0.1089	0.33
0	1	0	0.289	-0.289	0.083521	0.289
1	1	1	0.596	0.404	0.163216	0.404
						0.2855

Here's how we calculate RMS.

1. Figure out the error (desired output value minus actual output value). The absolute size of this number tells us how wrong the network is for this pattern.
2. Square this number, which ensures the result is positive.
3. Take the square root, which brings us back to the original error.
4. Take the mean of these numbers. The mean of (0.119, 0.33, 0.289, 0.404) is 0.2855

So the RMS Error for my network after 1000 sweeps is 0.29. Given that output node values vary from 0 to 1, we can call this a 30% error, which is what we see (more or less!) in the Error Display window (Figure 9):

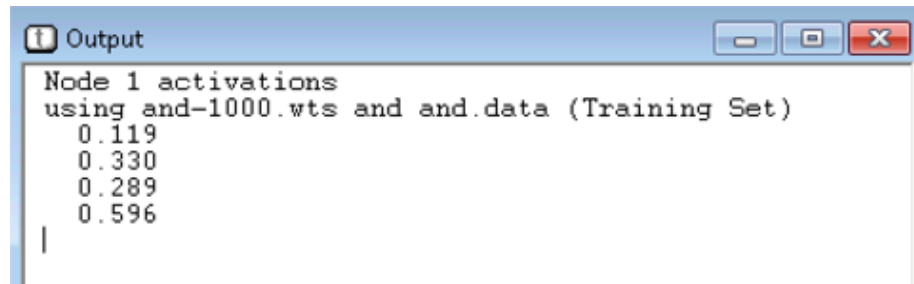


Figure 9: Tlearn Error Display

I'm going to keep training this network for 10000 sweeps (i.e. 10 times longer than before) and see what happens.

- (9) Follow along with me. Go to Network > Training Options and set the number of sweeps to 10000.
- (10) Now train the network using Network > Train the Network.

You'll see something like Figure 10.

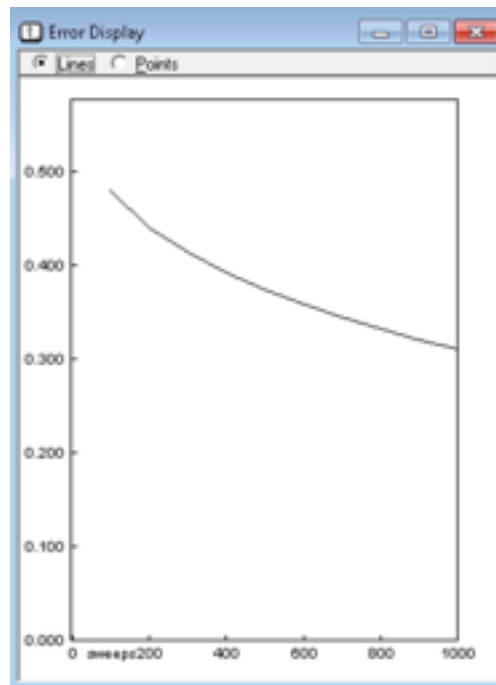


Figure 10: Tlearn Error Display

So error flattens at around 10%. In other words, the network is performing at a level of about 90% for the logical AND function... but only in the context of the learning parameters I've entered. These can be found under Network > Training Options. Two interesting parameters are 'learning rate' and 'momentum'.

Looks what happens if I increase the learning rate from 0.1 to 0.6 (Figure 11).

Now, the network is doing much better. Error falls off more quickly, and the overall performance at 10000 sweeps is more impressive.

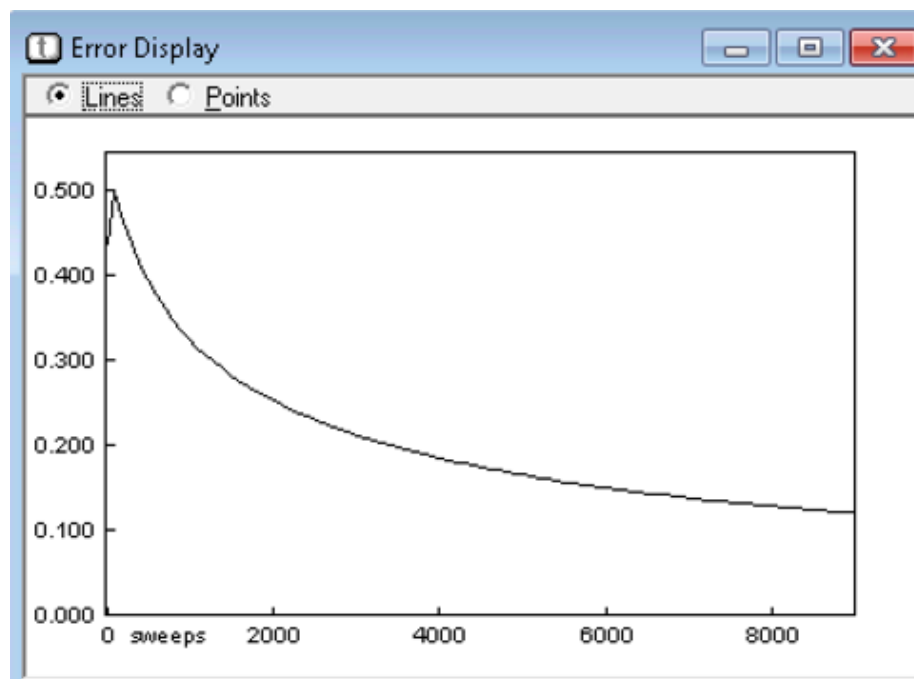


Figure 11: Tlearn Error Display

(11) Use the ‘Probe selected nodes’ feature. What do you notice about the patterns? Are they all learned equally well?

(12) In your journal, experiment with changing momentum and learning rate. How low can you get error after 10000 sweeps? Use a table to keep track of things. It might look something like Table 6.

Table 6: Exploring learning rate and momentum

Learning Rate	Momentum	Error per cent at 10k sweeps
0.2	0.2	
0.4	0.2	
0.6	0.2	
0.8	0.2	
1	0.2	
0.2	0.4	
0.4	0.4	
0.6	0.4	
And so on...	And so on...	And so on...

It would be useful to draw picture of the network with the weights written in. To do this, we look for the relevant .wts file (which is a text file openable by Notepad). Mine is called ‘and-10000.wts’. The file contains this:

```

NETWORK CONFIGURED BY TLEARN
# weights after 10000 sweeps
# WEIGHTS
# TO NODE 1
-9.0288238525
5.9597125053
5.9611163139
0.0000000000

```

These weights are all to the output node. The fifth line, -9.03, is the bias node—obviously, it's being strongly inhibitive. Lines six and seven are the connection weights between the two input nodes (both about 6), while the eighth line is the weighted connection from the output node to itself (there isn't one, so this is reported as 0).

Figure 12 shows my network with a learning rate of 0.6, momentum 0, and after 10k sweeps.

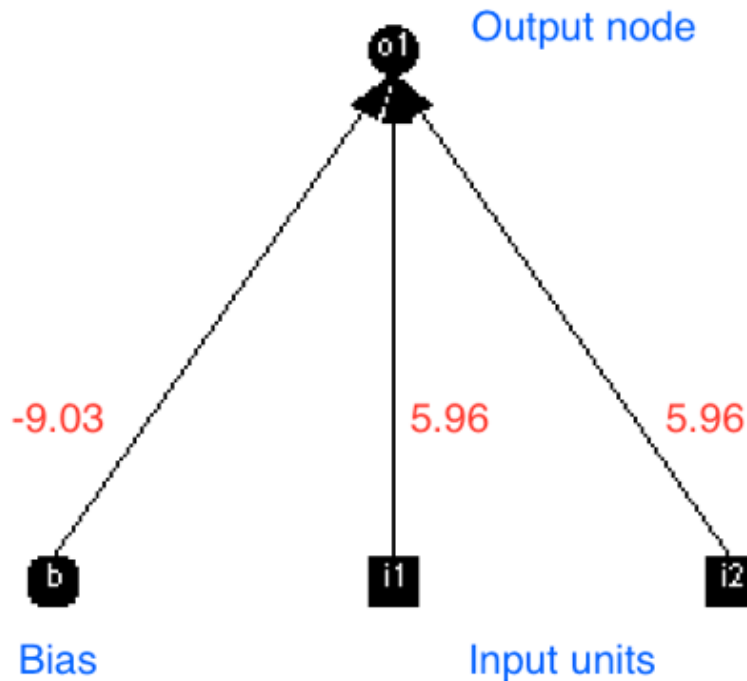


Figure 12: Network Diagram

- (13) Draw a picture of your most successful network from Table 6 and include the connection weights.

There are two other important functions to look at. Remember, we're trying to model basic computing functions using neural networks. If we can do this, we're on the road to building a mind inside a computer. The first is called OR (Table ??). The second is called exclusive OR, or XOR (Table ??).

Table 7: The OR function {tbl:or}

Input 1	Input 2	Output (OR function)
0	0	0
1	0	1
0	1	1
1	1	1

Table 8: The XOR function {tbl:xor}

Input 1	Input 2	Output (XOR function)
0	0	0
1	0	1
0	1	1
1	1	0

- (14) Quit and relaunch Tlearn (to clear the decks, as it were). Now open up the OR project and run the simulation.

- (15) Does it learn successfully? Draw the most successful network with its weights.

- (16) Quit and relaunch Tlearn (to clear the decks, as it were). Now open up the XOR project and run the simulation.

(17) Does it learn successfully? Draw the most successful network with its weights.

5 Versions

This document is available in [standard PDF](#), [simplified Layout PDF](#), [Microsoft Word format](#) and [webpage](#).

5.1 Licence

This work is licensed under a [Creative Commons](#) Attribution-NonCommercial-NoDerivatives 4.0 International License. The dark theme is based on [Solarised](#) by Ethan Schoonover.

The header image featuring HAL from *2001: A Space Odyssey* was made by Pixabay [CC0](#), via [Wikimedia Commons](#).

<i>Date</i>	07/22/17 18:27:45
<i>Word count</i>	3922
<i>Computer</i>	Winnetou
<i>Doc ID</i>	dff3df720cb5023bc29b12d1694d6b1c
<i>Compiler</i>	Octavo
