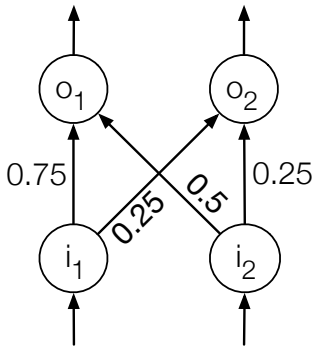
Tutorial 1
Introduction to Connectionism

1 Simple Example: Net1



1.1 Computing Output Activations and Error

1. Given the network on the right, determine by hand the output activations for the four input patterns in the table. Assume the logistic activation function (see table at the end of this tutorial, or use your calculator).

Pattern		Output_1	Output_2
0	0		
0	1		
1	0		
1	1		

Formulae

- The "generalized" Delta rule: $\Delta w_{ij} = \varepsilon\, \delta_i\, a_j$
  - For output nodes: $\delta_i = \sigma'(net_i)\,(t_i - a_i)$
  - For hidden nodes: $\delta_i = \sigma'(net_i)\, \sum_k \delta_k\, w_{ki}$

  (i,j,k: nodes; t: target; a: activation; ε: learning rate; net_i: input to node i; Δw: change to w; w_{ij}: weight from node j to node i—note the conventional ordering of indices!)

- The logistic or sigmoid activation function: $a_i = \sigma(net_i) = \dfrac{1}{1 + e^{-net_i}}$
  Its first derivative: $\sigma'(net_i) = a_i\,(1 - a_i)$

- Root mean square (RMS) error: $\sqrt{\dfrac{\sum_p (\vec{t} - \vec{a})^2}{p}}$ (p: number of patterns; $\vec{v}$: vector)

2. Given the following desired outputs, calculate the RMS error.

   (Remember that the dot product of two vectors is a scalar. It can be calculated by multiplying the corresponding terms in the two vectors and summing the results, e. g. if $\vec{a}$ is [1 4] and $\vec{b}$ is [2 0], the dot product $a \cdot b$ is $1 \times 2 + 4 \times 0 = 2$.)

Pattern		Output (from p. 1)		Desired	
i_1	i_2	o_1	o_2	o_1	o_2
0	0			0	0
0	1			1	1
1	0			1	1
1	1			2	2

1.2 Getting Tlearn

1. Create a directory for the Connectionism tutorials in your home directory.
2. Download Tlearn to this directory and unzip it:
   ftp.crl.ucsd.edu/pub/neuralnets/tlearn/wintlrn1.0.3.zip
3. Open a terminal (Applications → Terminal or System Tools → Konsole).
4. Using *cd*, change to the directory where Tlearn is located.
5. Start Tlearn with
   `$ wine tlearn.exe &`
   (This may take a while)

1.3 Opening and Displaying a Network

1. Download the archive `Net1.zip` and unzip it.
2. Open the project by clicking Network (not File) → Open Project and selecting `net1` (without extension or ending in `.prj`).
3. In the menu, select Displays → Network Architecture to examine the network graphically.
4. At the top of the window, uncheck Bias if checked.

This is the network for which you just computed the activations and error values.

1.4 Network Architecture, Input and Teach Files

The network's architecture is specified in `net1.cf` and shown when the project is opened. Note that the order of the lines in the file is fixed. What do the following lines mean? Look the answers up in the manual if needed (available from *ftp.crl.ucsd.edu/pub/neuralnets/tlearn/TlearnManual.pdf*).

```
1   NODES:
2   nodes = 2              ....................................
3   inputs = 2
4   outputs = 2
5   output nodes are 1-2   ....................................
6   CONNECTIONS:
7   groups = 0             (Ignore)
8   1-2 from i1-i2         ....................................
9   SPECIAL:
11  weight limit = 4.00    ....................................
```

In addition, when you open a project, Tlearn also opens two other files. These are:

- `net1.data`: the input patterns
  (the number of columns must match line 3 in the `.cf` file)
- `net1.teach`: the desired output patterns
  (the number of columns must match line 4 in the `.cf` file)

1.5 Weights and Output Activations

In this example, instead of training the network, we are going to use a set of prespecified weights and examine the output as if we had trained it and were about to test its performance on a second set of data.

1. Open `net1.wts` to examine it (either in Tlearn via the File menu or in the shell using *less* or your favorite editor).
   a) How many sections does the file contain after the header and why? (Think about networks with hidden units)

   b) How are the lines in each section ordered?

   c) What do zero values mean?

2. Now load the weights file by clicking on Network → Testing Options, then selecting Network Weights file, Earlier one, double-clicking in the box on the right, and choosing `net1.wts`.

3. To obtain the results produced by the network, select Network → Verify the network has learned. The output window opens: Do the values shown correspond to those you computed by hand?

2 Learning Boolean Functions

1. Create a directory named `And` containing a new Tlearn project called `and` (New Project → File name).

2. Create the network (two input nodes connected to one output node) in the architecture configuration file.

3. Make sure
   - All three sections are present in the configuration file (`NODES`, `CONNECTIONS`, and `SPECIAL`, each ending in a colon).
   - The first line under `CONNECTIONS` is `groups = 0` (which you can ignore).

4. Add a connection between the output node and the bias node (node number 0).

5. Create the data and teach files.

6. Set the training options (Network → Training Options) to: 1000 sweeps, seed with 1, train randomly without replacement (open more... at the bottom of the dialog box), learning rate 0.1, momentum 0.

7. Train the network, then test it using Verify the network has learned. The output should be similar to `0.147, 0.306, 0.325, 0.551`.

8. Has the network learned successfully? Explain the criteria for your decision.

9. Now set the network to train for 10000 sweeps, but specify training should stop if the RMS error falls below 0.25 (see Plunkett and Elman, 1997, *Exercises...*, pp. 64–65 for the logic behind this value). How long does it take for the network to learn, given this criterion?

10. Assume the network is reset and the random weights are:
    $w_{1,0} = -0.50$ (bias to output node)
    $w_{1,i1} = -0.75$
    $w_{1,i2} = 0.40$
    Go through the steps of gradient descent for 00, 01, 10, 11, 00, 01, assuming a learning rate of 10 to make faster changes.

11. Make a copy of your project and change the learning patterns to learn `Or`. How long does the network take to learn the solution to this problem?

12. Now try `Xor`. Can you make the network learn this problem?

3 Networks with Hidden Layers

1. Make a copy of `Or` called `Xor`, modify the network configuration file to add two hidden units, fully connected to the input nodes, and adapt `xor.teach`. Then train the network. How long does it take for the network to learn this function?

2. Open the error display. Do you notice anything special about how the network finds a solution to the problem over time?

3. On the back of this sheet, do the math for one complete training epoch by hand, assuming the values given below to start with and concentrating on how we apportion error to the each of the hidden units.

$w_{1,0}$	0.75	$w_{1,i1}$	-0.40	$w_{3,1}$	-0.25
$w_{2,0}$	-0.60	$w_{2,i1}$	0.35	$w_{3,2}$	0.45
$w_{3,0}$	-0.65	$w_{1,i2}$	-0.85		
		$w_{2,i2}$	0.15		

4. Examine the connection weights after training the network successfully. Draw the network in the space below and make a table of the hidden unit activations for each input pattern.

5. How can you describe the solution the network has found? What is the role of the hidden units in learning the function?

The Logistic function

Input	Activation	Derivative
-5.99	0.0025	0.0025
-5.29	0.005	0.005
-4.60	0.01	0.01
-3.89	0.02	0.02
-3.48	0.03	0.03
-3.18	0.04	0.04
-2.94	0.05	0.05
-2.51	0.075	0.07
-2.20	0.1	0.09
-1.95	0.125	0.11
-1.73	0.15	0.13
-1.55	0.175	0.14
-1.39	0.2	0.16
-1.10	0.25	0.19
-0.85	0.3	0.21
-0.62	0.35	0.2275
-0.41	0.4	0.24
-0.20	0.45	0.2475
0	0.5	0.25
0.20	0.55	0.2475
0.41	0.6	0.24
0.62	0.65	0.23
0.85	0.7	0.21
1.10	0.75	0.19
1.39	0.8	0.16
1.55	0.825	0.14
1.73	0.85	0.13
1.95	0.875	0.11
2.20	0.9	0.09
2.51	0.925	0.07
2.94	0.95	0.05
3.18	0.96	0.04
3.48	0.97	0.03
3.89	0.98	0.02
4.60	0.99	0.01
5.29	0.995	0.005
5.99	0.9975	0.0025