CS 182 Spring 2004. Assignment 2 Solution.
TLEARN software exercises (1)

The solutions here are model answers only. There are a number of equally good answers to these questions.

Part 1: Designing the logical AND function [10 points]

The first task is the logical AND function: AND(0,0)=AND(0,1)=AND(1,0)=0; AND(1,1)=1.

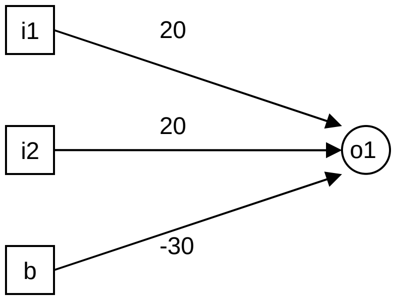
1. Show the network architecture (weights, output function) you designed and its output on the 4 AND input patterns. A printed or hand-drawn network architecture with weights indicated is recommended.

Sigmoid activation function

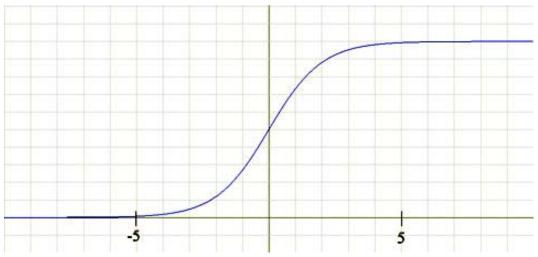
```
NODES:
nodes = 1
inputs = 2
outputs = 1
output node is 1
CONNECTIONS:
groups = 0
1 from i1-i2 = 20 & 20 fixed
1 from 0 = -30 & -30 fixed
SPECIAL:
selected = 1
weight_limit = 1.00
```

Linear activation function

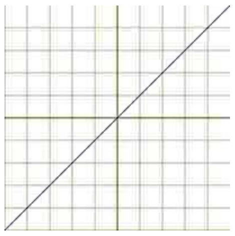
```
NODES:
nodes = 1
inputs = 2
outputs = 1
output node is 1
CONNECTIONS:
groups = 0
1 from i1-i2 = 20 & 20 fixed
1 from 0 = -30 & -30 fixed
SPECIAL:
linear = 1
selected = 1
weight_limit = 1.00
```



Sigmoid activation function



Linear activation function



Note that you would have to set the weights fairly large to get the outputs close to 0 and 1 because of the way the sigmoid function works. If the net input falls between the range of approximately -5 to 5, the output will stay in the middle region of the sigmoid, hovering at around 0.3 - 0.7

Sigmoid activation function

```
Output activations
using and-1.wts and and.data
(Training Set)
  0.000
  0.000
  0.000
  1.000
```

Linear activation function

```
Output activations
using and-1.wts and and.data
(Training Set)
-30.000
-10.000
-10.000
 10.000
```

2. Show how the network works, by illustrating the calculations that produce the output for each pattern.

For sigmoid, output = $\frac{1}{1+e^{-net_i}}$. For linear, output = net_i

i1	i2	net_i	output (sigmoid)	output (linear)
0	0	-30	1/(1+e^30) = 0.000	-30
0	1	-10	1/(1+e^10) = 0.000	-10
1	0	-10	1/(1+e^10) = 0.000	-10
1	1	10	1/(1+e^-10) = 1.000	10

Part 2: Learning the logical AND function

2a. Learning without hidden nodes [20 points]

1. Briefly describe the learning criterion you used.

tlearn, by default, calculates Root Mean Squared (RMS) error and that is the error plotted in the error display.

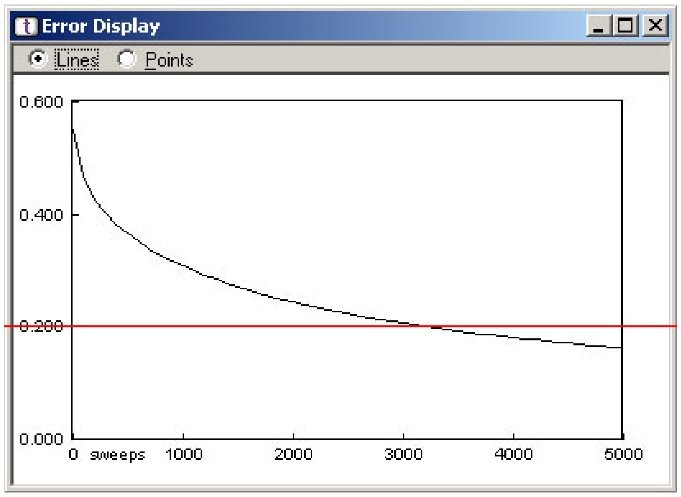
For this assignment, RMS < 0.200 would be a good enough criterion.

Usually the error would not go all the way to 0.000, and in fact, in most applications it is undesirable to train till the network reaches zero error. Over-training would reduce the network's ability to generalize what it learned to new data.

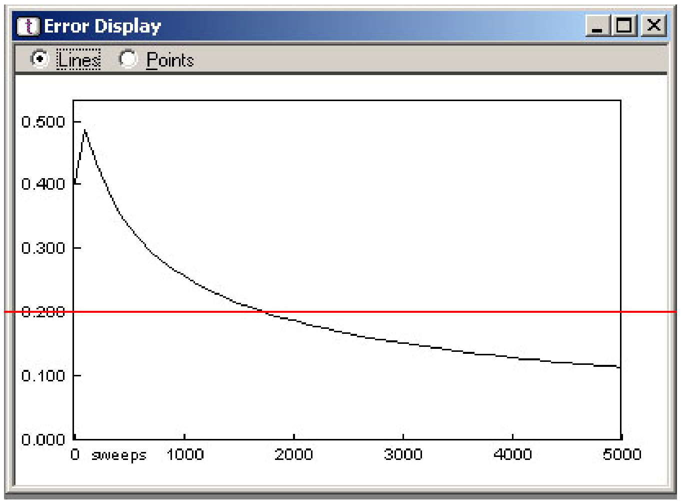
2. Turn in a record of the (hand-recorded) parameters that you tried as well as an account of what happened. Include an example solution.

Here is a typical error curve and the solution:

# Training Sweeps = 5000; Learning Rate = 0.1; Momentum = 0; Train Sequentially; Sigmoid



```
NETWORK CONFIGURED BY TLEARN
# weights after 5000 sweeps
# WEIGHTS
# TO NODE 1
-4.4863400459
2.9141252041
2.9166762829
0.0000000000

Output activations
using and-5000.wts and
and.data (Training Set)
   0.011
   0.172
   0.172
   0.793
```



```
NETWORK CONFIGURED BY TLEARN #
weights after 5000 sweeps #
WEIGHTS
# TO NODE 1
-5.7468814850
3.7612683773
3.7623276711 0.0000000000

Output activations
using and-5000.wts and
and.data (Training Set)
   0.003
   0.121
   0.121
   0.855
```

Adding Momentum (in the above picture, momentum = 0.5) or increasing the learning rate generally make the error goes down faster.

3. For what range of settings does the network reliably learn the AND function?

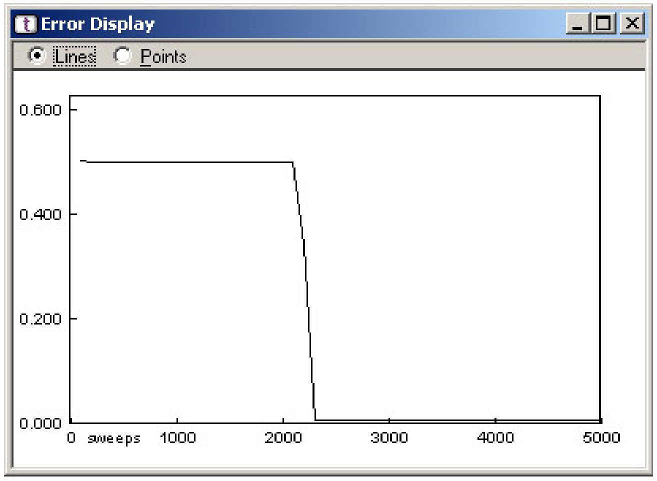
The network should reliably learn the AND function for any learning rate between 0 and 1.

In this case it would even learn reliably at a learning rate of 10, given that the momentum is very small (close to 0). This will not always be the case for other networks, however.

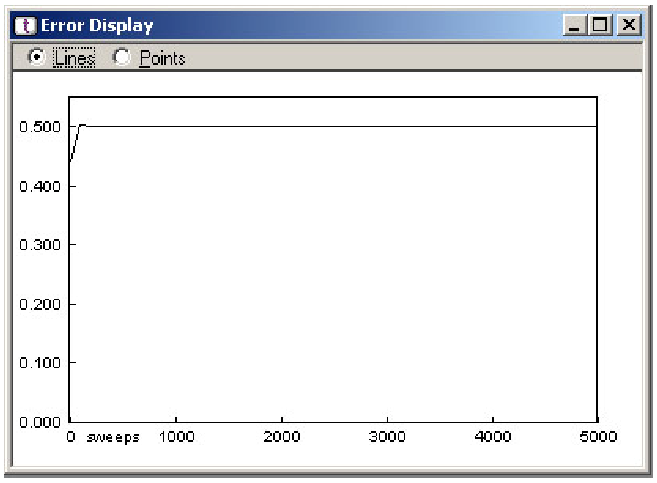
4. For what range of settings does the network learn about 75% of the time?

As you make the learning rate and momentum bigger and bigger, the learning starts to misbehave.

At learning rate = 10, momentum = 0.8, the error bumps around at 0.5 for a long while before going down to almost 0 (top picture). At learning rate = 10, momentum = 1, the network fails to learn completely (bottom picture).

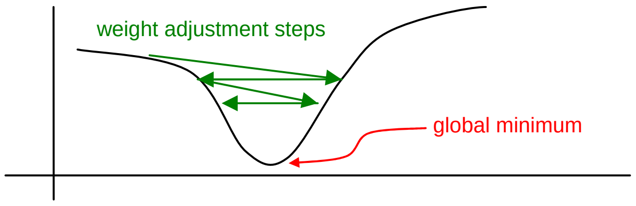


```
NETWORK CONFIGURED BY TLEARN #
weights after 5000 sweeps #
WEIGHTS # TO NODE 1 -
1535.9235839844
1385.2757568359
1464.3466796875 0.0000000000

Output activations
using and-5000.wts and
and.data (Training Set)
  0.000
  0.003
  0.004
  0.996
```



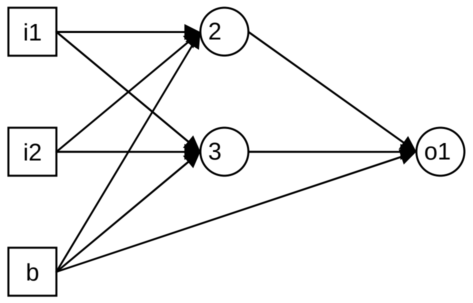
```
NETWORK CONFIGURED BY TLEARN
# weights after 5000 sweeps
# WEIGHTS
# TO NODE 1
-6933.8208007813
-561.5523681641
1494.9118652344
0.0000000000

Output activations
using and-5000.wts and
and.data (Training Set)
  0.000
  0.000
  0.000
  0.000
```

This indicates that in the process of minimizing the error of the network, the weight adjustment steps are too big and the steps jump around without finding its way down to the local (or global) minimum in the error gradient.



2b. Learning with hidden nodes [20 points]

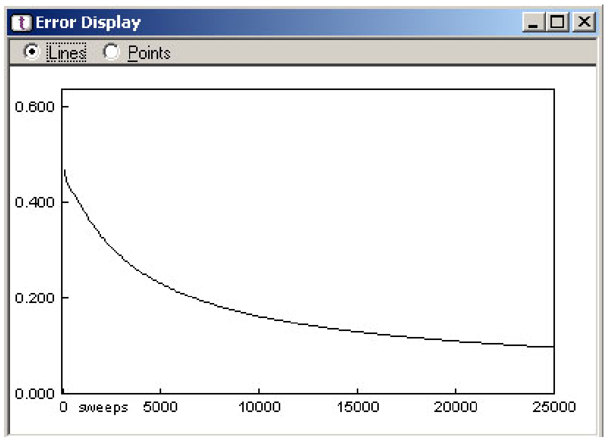
The network should look like this:

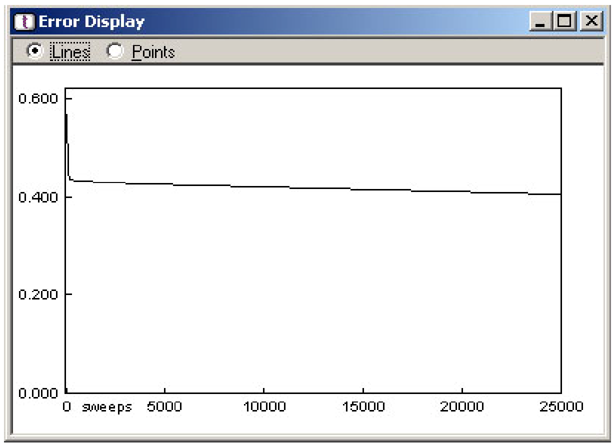


```
NODES:
nodes = 3
inputs = 2
outputs = 1
output node is 1
CONNECTIONS:
groups = 0
2-3 from i1-i2
1 from 2-3
1-3 from 0
SPECIAL:
selected = 1
weight_limit = 1.00
```

1. Once again, turn in a record of the (hand-recorded) parameters that you tried, a general account of what happened, and an example solution.

# Training Sweeps = 25000; Learning Rate = 0.5; Momentum = 0; Train Sequentially; Sigmoid





Both of the error curves above result from the same learning parameters. The addition of the two hidden nodes (6 connections) greatly increase the dimension of the weight space. As a result, this network learns much more slowly, but adding momentum helps. While there are no local minima, there are some plateau in the error space which leads to the flattening out of the error curve on the right hand side. The momentum helps propel the weight changes out of the plateau to the global minimum.

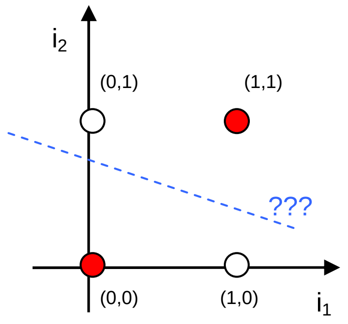
It trains fairly consistently at a learning rate of 0.5, momentum of 0.2 and 30000 sweeps. Like before, there is a trade-off between training time and learning rate – on the one hand, increasing learning rate (and momentum) drives the error down faster usually (reducing the number of training sweeps needed), increasing them too much can lead to the network not converging to the minima in the error gradient.

2. How much does this new network help?

This network in fact does not help in learning the AND function. The extra hidden nodes make learning more difficult.

Part 3: The logical SAME function [50 points]

The second task is the logical SAME function: SAME(0,0)= SAME (1,1)=1; SAME(0,1)= SAME(1,0)=0. Try to repeat the experiments involving (a) designed weights as in Part 1 and (b) learning as in Part 2, without and with two extra hidden units.



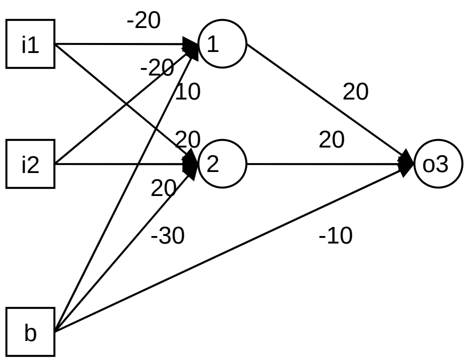
In lecture we learned that the XOR function is not linearly separable and therefore cannot be learned by a perceptron.

Similarly, the SAME function is not linearly separable. There is no way to draw a straight line on the graph to separate { (0,0), (1,1) } from { (0,1), (1,0) }.

Therefore it is not feasible to hand-set the weights to a single-layer perceptron to make it do the SAME function.

There are a number of ways to hand set the weights for the network below to make it perform the SAME function. One way is to think of the SAME function as performing ((i1 AND i2) OR (not i1 AND not i2)).

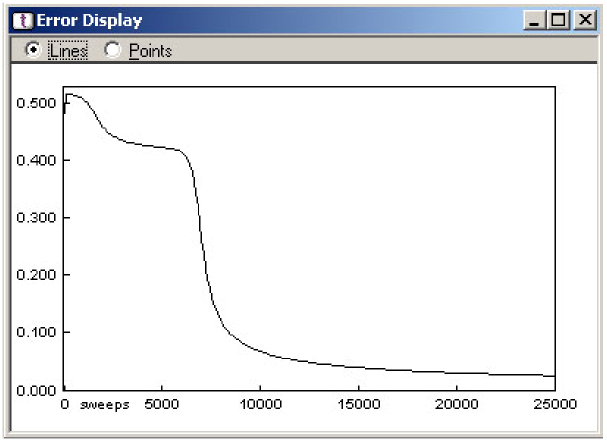
Essentially, node1 in the diagram below recognizes when both i1 and i2 are 0's, and node2 recognizes when both i1 and i2 are 1's. The output is then just a OR function of the outputs of node1 and node2.

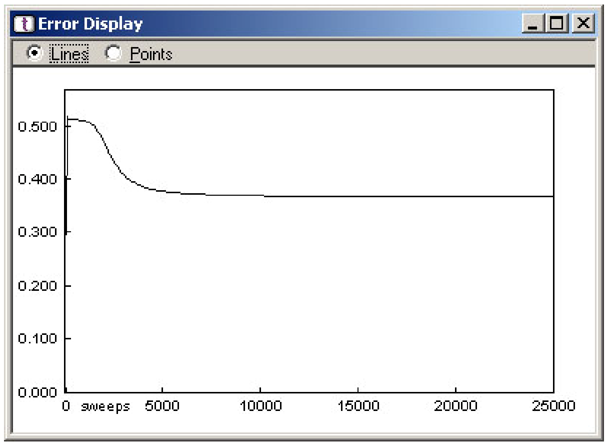


```
NODES:
nodes = 3
inputs = 2
outputs = 1
output node is 3
CONNECTIONS:
groups = 0
1 from i1-i2 = -20 & -20 fixed
2 from i1-i2 = 20 & 20 fixed
3 from 1-2 = 20 & 20 fixed
1 from 0 = 10 & 10 fixed
2 from 0 = -30 & -30 fixed
3 from 0 = -10 & -10 fixed
SPECIAL:
selected = 1-3
weight_limit = 1.00
```

Note: One of the students pointed out (and I noticed while creating the solution) that it does matter which node is the output node. The same configuration does not seem to work when the output node is designated node1 and the hidden nodes are node2 and node3.

1. Turn in a record of the (hand-recorded) parameters that you tried, a general account of what happened, and an example solution.



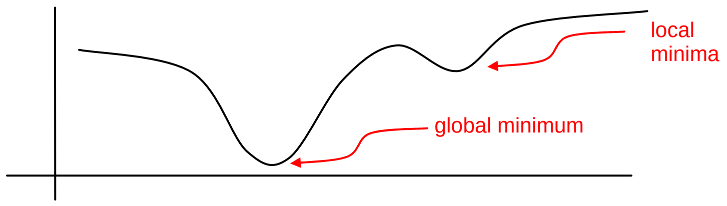


While using momentum helps, there are learning trials that will fail regardless. The error curve flattens out (at somewhere around 0.35).

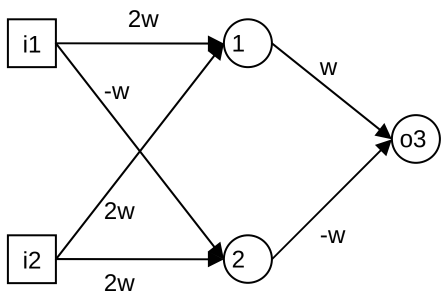
2. How do the results differ in this case? Explain why.

A local minima refers to a local "dip" in the error gradient that you can't get out of without first climbing upwards. The learning in a neural network proceeds by strictly minimizing the error (i.e. going downwards along the error gradient), so it will never try to climb up and out of the local dip.

In the picture below, if the random initialization of the network happens to fall on a point to the right hand side of the local minima, the learning will drive the network into the local minima. If the local minima is "deep enough", even the extra momentum factor cannot help the network get out of it.



(Rumelhart et al., 1986) has shown that a local minima exists in the XOR network with two hidden units.



A local minima occurs when weights are at this configuration.

To simplify the diagram the bias node is omitted.

Node1 has a threshold of m, Node2 has a threshold of 0, and Node3 has a threshold of 0.

The details of the proof is not crucial to this assignment. The key point is that whereas a higher learning rate and momentum can help the AND network (with 2 hidden nodes) learn faster, in the SAME network (which is just the negation of XOR), because of the local minima, there are certain cases where the learning rate and momentum will not help the situation.