Biological Neural Networks

Neural networks are inspired by our brains.

The human brain has about 10^{11} neurons and 10^{14} synapses.

A neuron consists of a soma (cell body), axons (sends signals), and dendrites (receives signals).

A synapse connects an axon to a dendrite.

Given a signal, a synapse might increase (excite) or decrease (inhibit) electrical potential. A neuron fires when its electrical potential reaches a threshold.

Learning might occur by changes to synapses.

Artificial Neural Networks

An (artificial) neural network consists of units, connections, and weights. Inputs and outputs are numeric.

Biological NN	Artificial NN
soma	unit
axon, dendrite	connection
synapse	weight
potential	weighted sum
threshold	bias weight
signal	activation

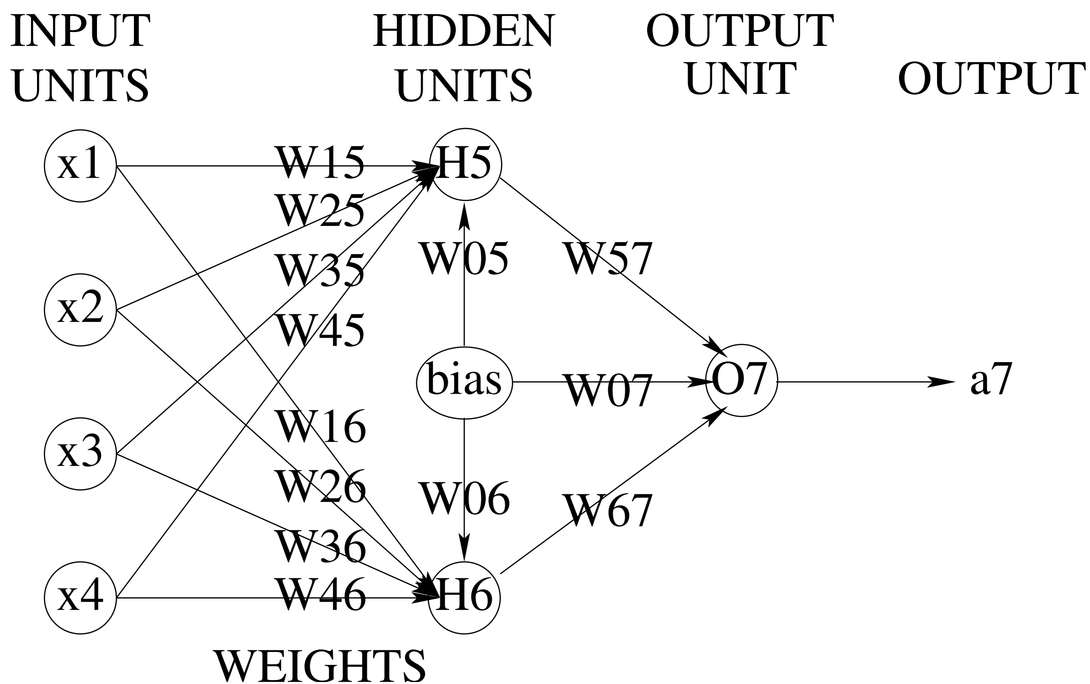
A typical unit j inputs $o_{j1}, o_{j2}, \ldots$ from units $j1$, $j2$, …, then performs a weighted sum

$$net_j = w_{j,0} + \Sigma\, w_{j,i}\; o_i$$

and outputs $o_j = a(net_j)$, where a is an activation fn.

In a typical ANN, *input units* store the inputs, *hidden units* transform the inputs into an internal numeric vector, and an *output unit* transforms the hidden values into the prediction.

An ANN is a function $o(\mathbf{w}, \mathbf{x})$, where $\mathbf{x}$ is an example and $\mathbf{w}$ is a set of weights. *Learning is finding values for* $\mathbf{w}$ *that minimizes error or loss over a dataset.*



How to Understand Most Neural Networks

$o = o(\mathbf{w}, \mathbf{x})$	How is the prediction o computed? How is o mapped to positive/negative class?
$(\mathbf{x}, t)$	A labelled example has a target value t.
$E(t, o)$	How is the error computed from t and o?
$\dfrac{\partial E(t,o)}{\partial \mathbf{w}}$	What is the *gradient*, the partial derivatives of the error wrt the weights? E and o must be continuous and differentiable.
$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$	How are the weights updated? How is $\Delta\mathbf{w}$ computed?

Initialization, Updating Details, Stopping Criterion

Example: Perceptron with a Margin

$o((b, \mathbf{w}), \mathbf{x}) = b + \mathbf{w}\cdot\mathbf{x} = b + \sum w_i x_i$

b is called the bias weight (the book uses w_0)

$t \in \{-1, 1\}$, $E(t, o) = \max(0, 1 - t * o)$

$$\frac{\partial E(t,o)}{\partial w_i} = -t * x_i \text{ and } \frac{\partial E(t,o)}{\partial b} = -t \text{ if } E > 0$$

$$\Delta w_i = -\eta \frac{\partial E(t,o)}{\partial w_i} \qquad \Delta b = -\eta \frac{\partial E(t,o)}{\partial w_i}$$

where $\eta > 0$ is the learning rate.

Initialize all weights to 0. Stop after zero error or epoch limit. Epoch = one pass over the examples.

Gradient Descent Algorithm
(incremental learning)

GRADIENT-DESCENT-INCREMENTAL(D)

1. initialize $\mathbf{w}$
2. **while** stopping criterion is false
3.   **for** each $(\mathbf{x}_j, t_j) \in D$
4.   **for** each $\Delta w_i \in \Delta\mathbf{w}$
5. $\Delta w_i \leftarrow -\eta * \partial E(t_j, o(\mathbf{w}, \mathbf{x}_j))/\partial w_i$
6.   **for** each $w_i \in \mathbf{w}$
7. $w_i \leftarrow w_i + \Delta w_i$
8. **return w**

Gradient Descent Algorithm
(batch learning)

GRADIENT-DESCENT-BATCH(D)

1. initialize $\mathbf{w}$
2. **while** stopping criterion is false
3.   **for** each $\Delta w_i \in \Delta\mathbf{w}$
4. $\Delta w_i \leftarrow 0$
5.   **for** each $(\mathbf{x}_j, t_j) \in D$
6.   **for** each $\Delta w_i \in \Delta\mathbf{w}$
7. $\Delta w_i \leftarrow \Delta w_i - \eta * \partial E(t_j, o(\mathbf{w}, \mathbf{x}_j))/\partial w_i$
8.   **for** each $w_i \in \mathbf{w}$
9. $w_i \leftarrow w_i + \Delta w_i$
10. **return w**

Linear Learning Algorithms

In linear learning, we compute o by:
$o((b, \mathbf{w}), \mathbf{x}) = b + \mathbf{w} \cdot \mathbf{x} = b + \sum w_i x_i$
and usually predict pos/neg by $o > 0$ and $o < 0$

LMS, Adaline, Widrow-Hoff (classification):
$t \in \{-1, 1\}$ and $E(t, o) = \max(0, 1 - t * o)^2/2$
$\Delta b = \eta * (t - o)$ and $\Delta w_i = \eta * x_i * (t - o)$ if $E > 0$

LMS, Adaline, Widrow-Hoff (regression):
$t \in \Re$ and $E(t, o) = (t - o)^2/2$
$\Delta b = \eta * (t - o)$ and $\Delta w_i = \eta * x_i * (t - o)$

Perceptron with a margin:
$t \in \{-1, 1\}$ and $E(t, o) = \max(0, 1 - t * o)$
$\Delta b = \eta * t$ and $\Delta w = \eta * t * x_i$ if $E > 0$

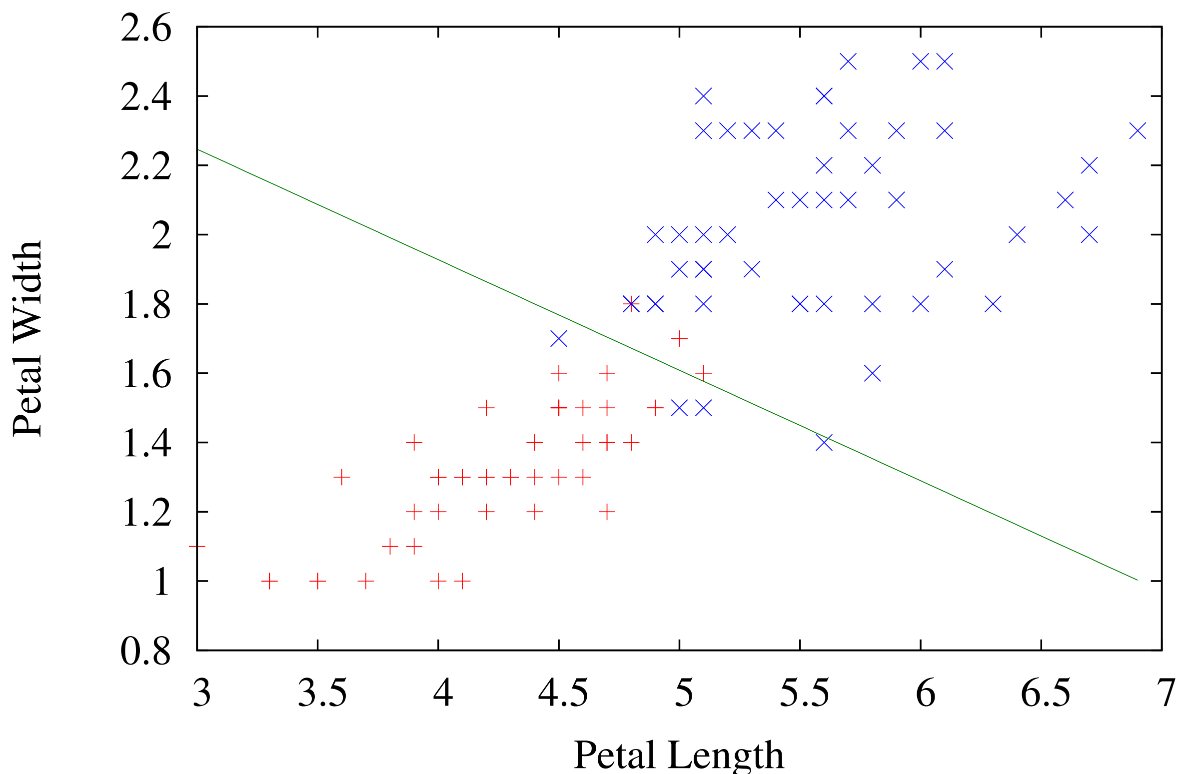
For low enough learning rates, these algorithms will be close to optimal error.

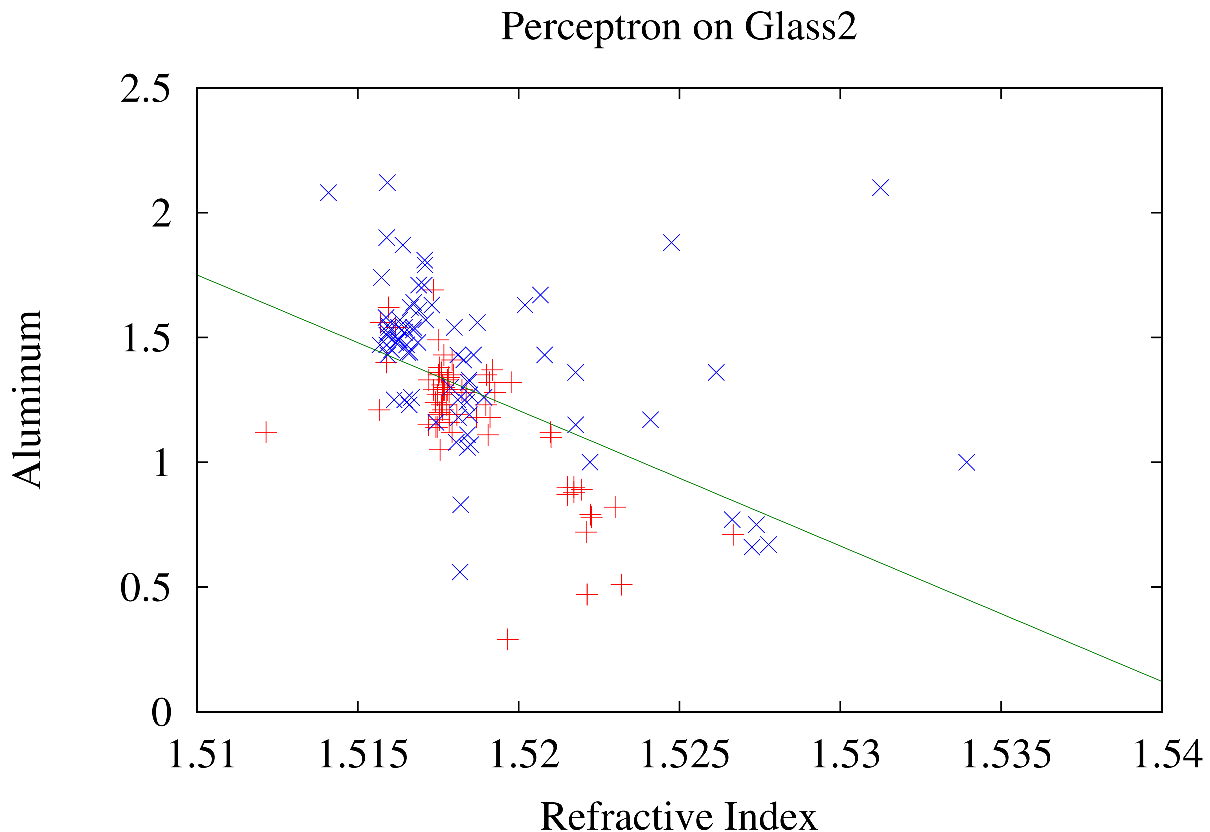
Optimizing E is not equivalent to minimizing classification errors except $E = 0$ implies no classification error.

When $E = 0$ is possible, we can derive a *mistake bound*, an upper limit on the number of mistakes during training.

Example of Perceptron Updating ($\eta = 1$)

Inputs							Weights				
x_1	x_2	x_3	x_4	t	o	E	b	w_1	w_2	w_3	w_4
							0	0	0	0	0
0	0	0	1	-1	0	1	-1	0	0	0	-1
1	1	1	0	1	-1	2	0	1	1	1	-1
1	1	1	1	1	2	0	0	1	1	1	-1
0	0	1	1	-1	0	1	-1	1	1	0	-2
0	0	0	0	1	-1	2	0	1	1	0	-2
0	1	0	1	-1	-1	0	0	1	1	0	-2
1	0	0	0	1	1	0	0	1	1	0	-2
1	0	1	1	1	-1	2	1	2	1	1	-1
0	1	0	0	-1	2	3	0	2	0	1	-1





Comments on Linear Learning

Learning is efficient if no very large weights. Efficient algorithms (not covered) if few relevant attributes out of many.

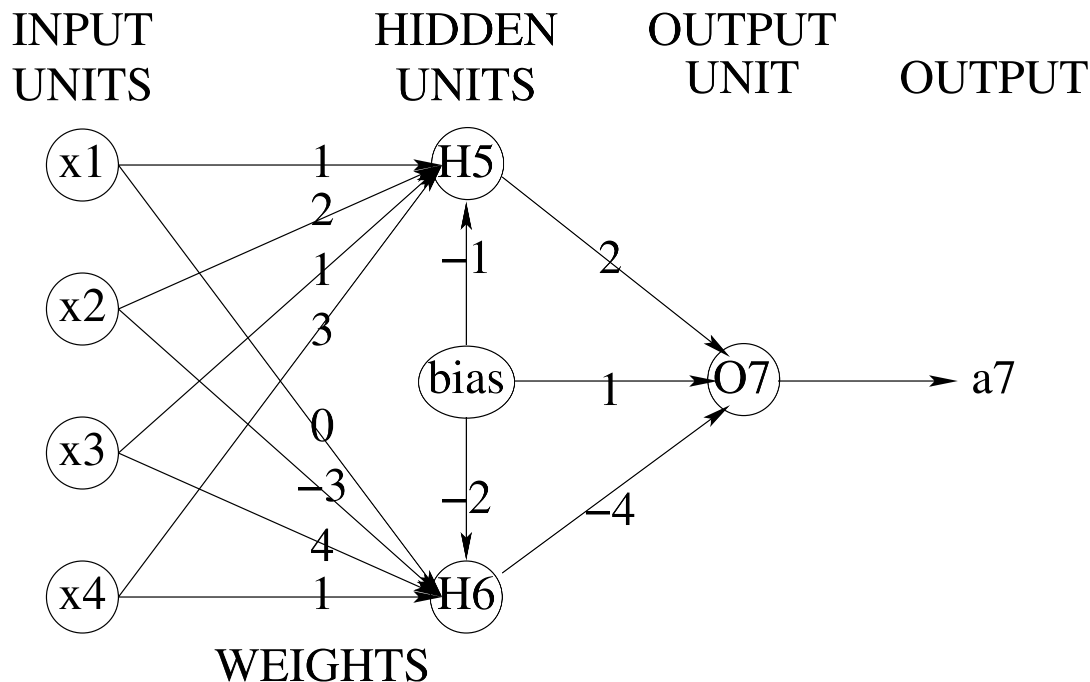
Can represent a single conjunction, disjunction, or k-out-of-n function.

Cannot learn exclusive OR.

Can only learn lines/hyperplanes.

Attributes are weighted independently.

Multilayer Networks



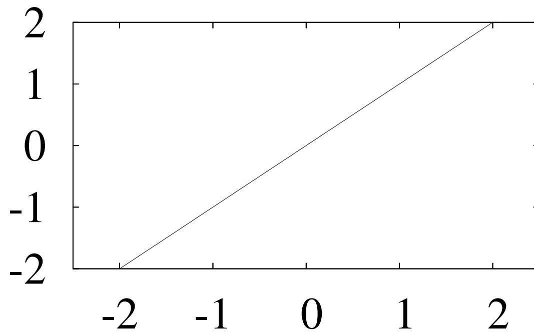
A feedforward network is an acyclic directed graph of *units*.

The input units provide the input values. All other units compute an output value from their inputs. Hidden units are internal.

If the hidden units compute a linear function, the network is equivalent to one without hidden units. Thus, the output of hidden units is produced by a nonlinear activation function. This is optional for output units.

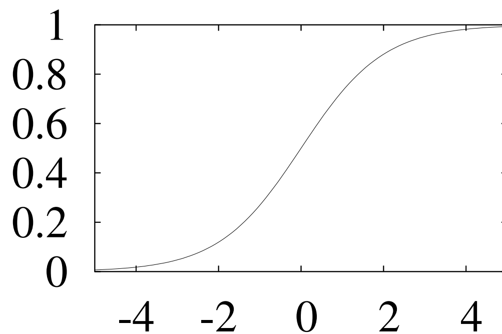
Activation Functions

Identity (Linear)



$$identity(x) = x$$

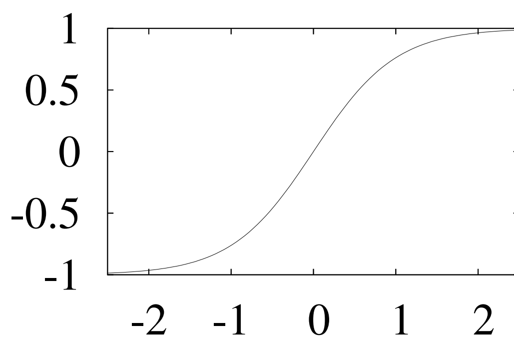
Sigmoid



$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

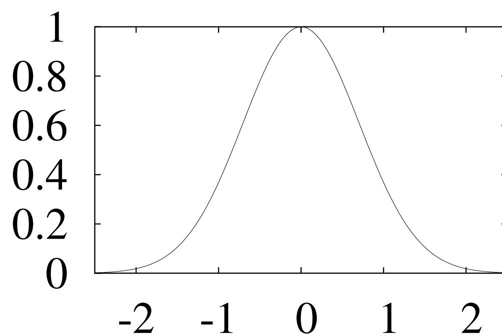
More Activation Functions

Tanh (Hypertangent)



$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Gaussian



$$gaussian(x) = e^{-x^2/\sigma^2}$$

Example

Suppose a feedforward neural network with n inputs,
m hidden units (tanh activation), and
l output units (linear activation).
v_{ji} is the weight from input i to hidden unit j.
w_{kj} is the weight from hidden unit j to output unit k.
Then the kth output o_k is:

$$w_{k,0} + \sum_{j=1}^{m} w_{k,j} \tanh(v_{j,0} + \sum_{i=1}^{n} v_{j,i}\, x_i)$$

If the error is: $\sum_{k=1}^{l} (t_k - o_k)^2$, we can find partial derivatives (backpropagation) and apply gradient descent.

Improving Efficiency

Neural networks can be slow to converge. There are several methods for speeding up convergence.

Adding *momentum*. Change line 4 of GD-BATCH to $\Delta w_i \leftarrow \alpha \Delta w_i$, where $0 \leq \alpha < 1$.

Quickprop and RPROP algorithms (see papers to present).

Conjugent gradient and other second derivative algorithms.

Improving Effectiveness

Weights cannot be initialized to 0. They need to be initialized to small random values.

Inputs need to be normalized (incremental) or standardized (batch).

ANNs can end up in local minina. One approach is many hidden units and some technique to avoid overfitting.

Overfitting can be avoided using weight decay (see book) or early stopping:
Remove a validation set from training exs.
Train neural network using training exs.
Choose weights that are best on validation set.

Properties of Neural Networks

Sufficiently complex neural networks can approximate any "reasonable" function. For example, we can construct an ANN for any boolean function.

- Any boolean function can be represented in CNF:
  A conjunction of clauses, each clause is a disjunction of literals, each literal is an attribute or its negation.
- Create a hidden unit for each clause.
- Output unit does an AND of the hidden units.

ANNs have a preference bias for smooth interpolation between data points.

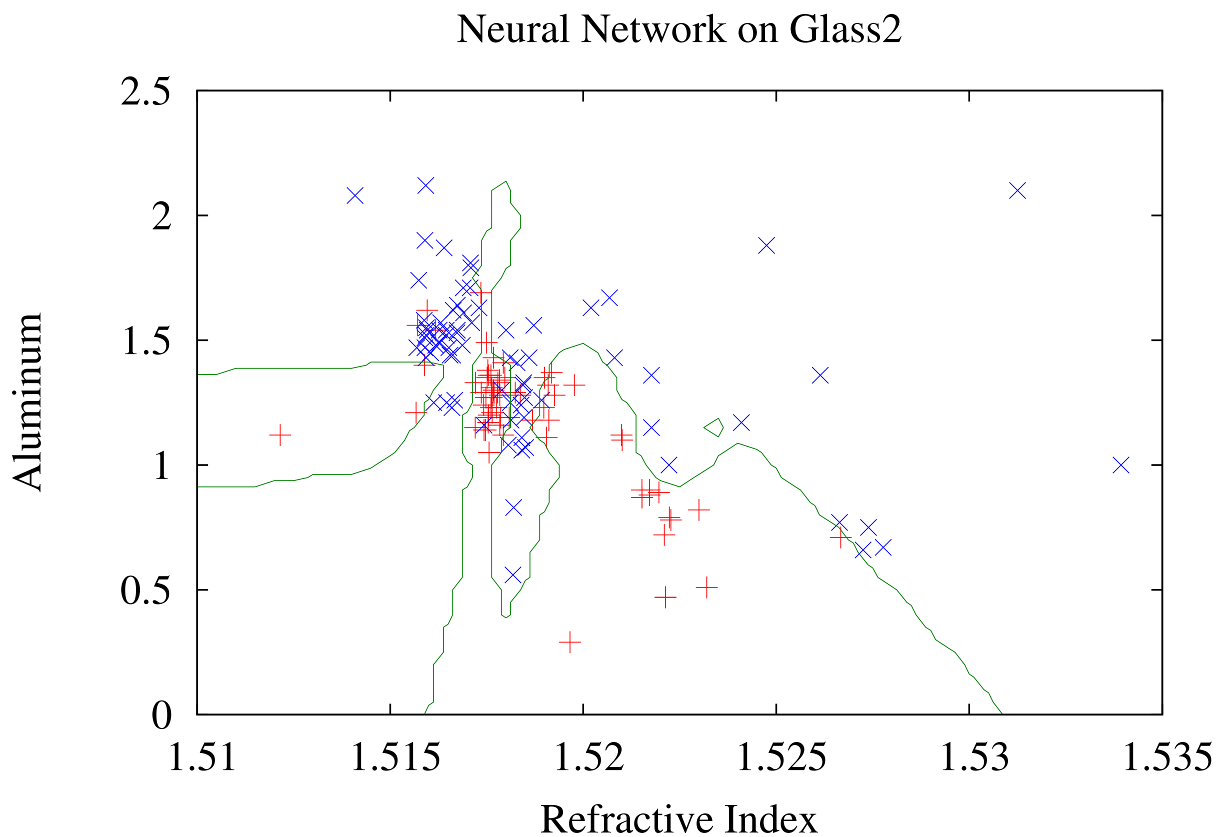
Example

The following example was generated by:

```
java
weka.classifiers.functions.MultilayerPerceptron
-t glass2.arff -T glass2.arff
-H 10 -N 10000 -L 0.1 -M .9
```

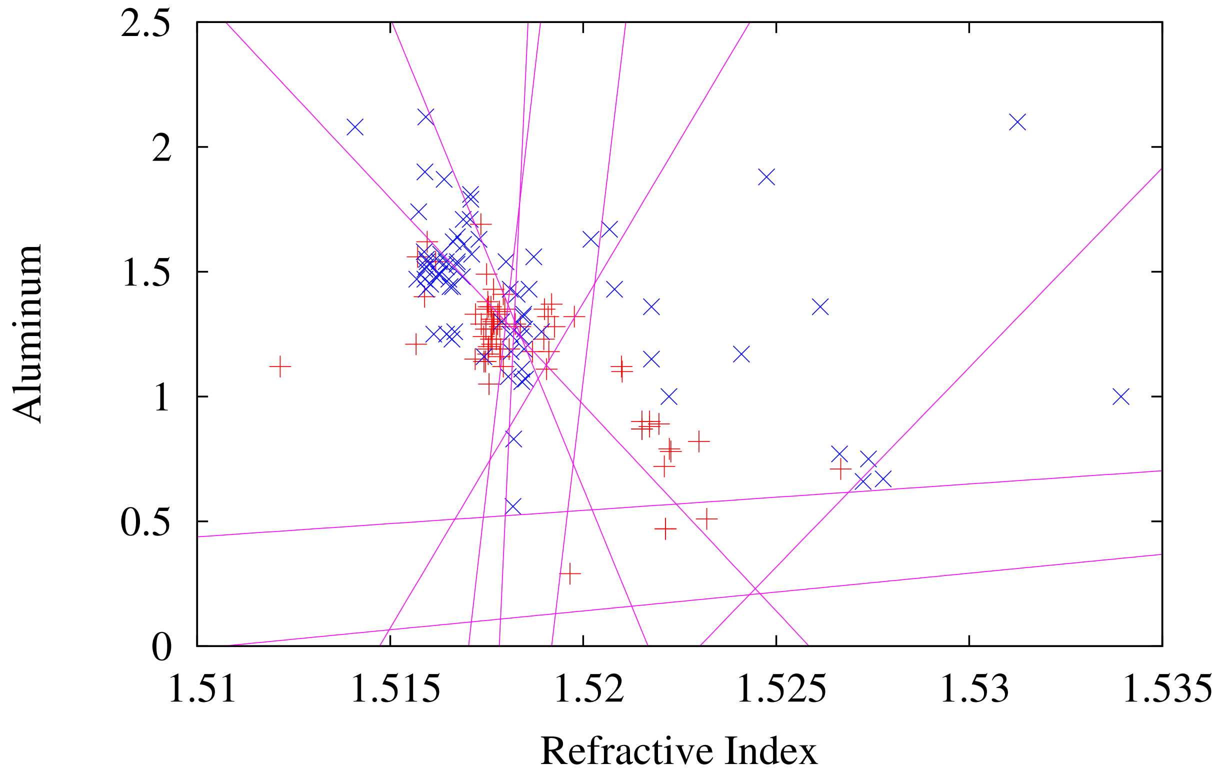
This command should all be on one line.

Decision Boundary of Neural Network



Centers of Hidden Units

Hidden Units on Glass2



Combined Picture

Output and Hidden Units on Glass2