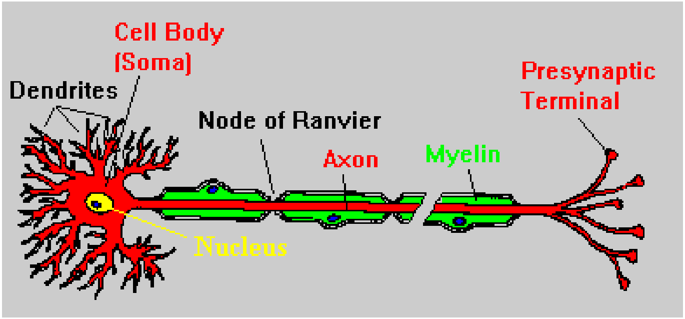
Statistical Databases and Registers with some statistical learning

a course in
Survey Methodology and Official Statistics
A neural network primer-Croft
A neural network primer-Abdi
(Pages in the book:389-401)

Department of Statistics
Stockholm University

December 2011

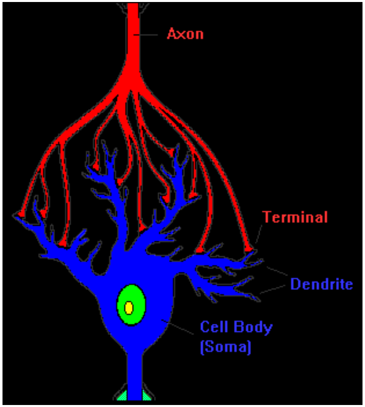
What is neurons?



The neuron network theory tries to mimic the human neuron

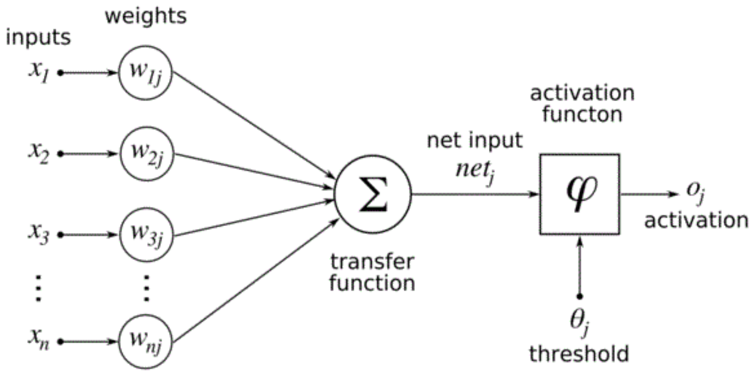
- Neurons have specialized extensions called dendrites (receivers), axons (cable) and terminals (senders). Dendrites bring information to the cell body and axons take information away from the cell body to the terminals.

What is neurons (cont)?



One neuron may have zero to many connections with another neuron. The more connections the stronger the signal.

Mathematical model of a neuron



Mathematical model of a neuron (cont)

The modeling of an artificial neuron has three basic components.

1. the synapses of the biological neuron are modeled as weights, w. Let's remember that the synapse of the biological neuron is the one which interconnects the neural network and gives the strength of the connection.

   For an artificial neuron the weight is a number. A negative weight reflects an inhibitory connection, while positive values designate excitatory connections.

2. all inputs, x, are modified by the weights and the result is summed. This activity is referred as a linear combination.

3. an activation function, ϕ, controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1.

Mathematical model of a neuron (cont)

The modeling of an artificial neuron has three basic components.

1. the synapses of the biological neuron are modeled as weights, w. Let's remember that the synapse of the biological neuron is the one which interconnects the neural network and gives the strength of the connection.

   For an artificial neuron the weight is a number. A negative weight reflects an inhibitory connection, while positive values designate excitatory connections.

2. all inputs, x, are modified by the weights and the result is summed. This activity is referred as a linear combination.
3. an activation function, ϕ, controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1.

Mathematical model of a neuron (cont)

The modeling of an artificial neuron has three basic components.

1. the synapses of the biological neuron are modeled as weights, w. Let's remember that the synapse of the biological neuron is the one which interconnects the neural network and gives the strength of the connection.

   For an artificial neuron the weight is a number. A negative weight reflects an inhibitory connection, while positive values designate excitatory connections.

2. all inputs, x, are modified by the weights and the result is summed. This activity is referred as a linear combination.
3. an activation function, ϕ, controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1.

Mathematical model of a neuron - the activity

For two artificial neurons k and j we define the following variables

$$\begin{aligned} x_j &= \text{signal/no-signal received from neuron } j \\ w_{kj} &= \text{connection weight from neuron } j \text{ to neuron } k \end{aligned}$$

The internal activity in the neuron is the linear combination

$$v_k = \sum_{j=1}^{p} w_{kj} x_j$$

and on the output v_k we have some activation function which gives us the decision $\phi(v_k)$.

Hence neuron k get inputs from the neurons $j \in \{1, 2, \ldots, p\}$.

Neuron k has an idle state that we model with a constant signal $x_0 = 1$ and a constant weight w_{k0}, so finally $v_k = \sum_{j=0}^{p} w_{kj} x_j$.

Mathematical model of a neuron - the activity

For two artificial neurons k and j we define the following variables

$$\begin{aligned} x_j &= \text{signal/no-signal received from neuron } j \\ w_{kj} &= \text{connection weight from neuron } j \text{ to neuron } k \end{aligned}$$

The internal activity in the neuron is the linear combination

$$v_k = \sum_{j=1}^{p} w_{kj} x_j$$

and on the output v_k we have some activation function which gives us the decision $\phi\left(v_k\right)$.

Hence neuron k get inputs from the neurons $j \in \{1, 2, \ldots, p\}$.

Neuron k has an idle state that we model with a constant signal $x_0 = 1$ and a constant weight w_{k0}, so finally $v_k = \sum_{j=0}^{p} w_{kj} x_j$.

Mathematical model of a neuron - the activity

For two artificial neurons k and j we define the following variables

$$\begin{aligned} x_j &= \text{signal/no-signal received from neuron } j \\ w_{kj} &= \text{connection weight from neuron } j \text{ to neuron } k \end{aligned}$$

The internal activity in the neuron is the linear combination

$$v_k = \sum_{j=1}^{p} w_{kj} x_j$$

and on the output v_k we have some activation function which gives us the decision $\phi\left(v_k\right)$.

Hence neuron k get inputs from the neurons $j \in \{1, 2, \ldots, p\}$.

Neuron k has an idle state that we model with a constant signal $x_0 = 1$ and a constant weight w_{k0}, so finally $v_k = \sum_{j=0}^{p} w_{kj} x_j$.

Mathematical model of a neuron - the activity

For two artificial neurons k and j we define the following variables

$$\begin{aligned} x_j &= \text{signal/no-signal received from neuron } j \\ w_{kj} &= \text{connection weight from neuron } j \text{ to neuron } k \end{aligned}$$

The internal activity in the neuron is the linear combination

$$v_k = \sum_{j=1}^{p} w_{kj} x_j$$

and on the output v_k we have some activation function which gives us the decision $\phi(v_k)$.

Hence neuron k get inputs from the neurons $j \in \{1, 2, \ldots, p\}$.

Neuron k has an idle state that we model with a constant signal $x_0 = 1$ and a constant weight w_{k0}, so finally $v_k = \sum_{j=0}^{p} w_{kj} x_j$.

Mathematical model of a neuron - the activation function

In general, there are three types of activation functions, $\phi(\cdot)$.

The first one is the treshold function

$$\phi(v) = \begin{cases} 1 & v \geq 0 \\ 0 & v < 0 \end{cases}$$

The second one is the piece-wise linear function e.g.

$$\phi(v) = \begin{cases} 1 & v \geq 0.5 \\ v & -0.5 \leq v < 0.5 \\ 0 & v < -0.5 \end{cases}$$

The third one is the sigmoid function e.g.

$$\phi(v) = \frac{1}{1 + e^{-v}}$$

(pictures here)

Mathematical model of a neuron - the activation function

In general, there are three types of activation functions, $\phi\left(\cdot\right)$.

The first one is the treshold function

$$\phi\left(v\right)=\begin{cases}1 & v\geq0\\ 0 & v<0\end{cases}$$

The second one is the piece-wise linear function e.g.

$$\phi\left(v\right)=\begin{cases}1 & v\geq0.5\\ v & -0.5\leq v<0.5\\ 0 & v<-0.5\end{cases}$$

The third one is the sigmoid function e.g.

$$\phi\left(v\right)=\frac{1}{1+e^{-v}}$$

(pictures here)

Mathematical model of a neuron - the activation function

In general, there are three types of activation functions, $\phi(\cdot)$.
The first one is the treshold function

$$\phi(v) = \begin{cases} 1 & v \geq 0 \\ 0 & v < 0 \end{cases}$$

The second one is the piece-wise linear function e.g.

$$\phi(v) = \begin{cases} 1 & v \geq 0.5 \\ v & -0.5 \leq v < 0.5 \\ 0 & v < -0.5 \end{cases}$$

The third one is the sigmoid function e.g.

$$\phi(v) = \frac{1}{1 + e^{-v}}$$

(pictures here)

Mathematical model of a neuron - the activation function

In general, there are three types of activation functions, $\phi(\cdot)$.
The first one is the treshold function

$$\phi(v) = \begin{cases} 1 & v \geq 0 \\ 0 & v < 0 \end{cases}$$

The second one is the piece-wise linear function e.g.

$$\phi(v) = \begin{cases} 1 & v \geq 0.5 \\ v & -0.5 \leq v < 0.5 \\ 0 & v < -0.5 \end{cases}$$

The third one is the sigmoid function e.g.

$$\phi(v) = \frac{1}{1 + e^{-v}}$$

(pictures here)

Example

We get the logical OR function from the choice:
$p = 2,\ w_0 = -0.5,\ w_1 = w_2 = 0.5$
and the treshold function

(picture here)

as activation function.

x_1	x_2	v	$\phi(v)$	$x_1 \vee x_2$
1	1	$-0.5 \times 1 + 0.5 \times 1 + 0.5 \times 1$	$\phi(0.5)$	1
1	0	$-0.5 \times 1 + 0.5 \times 1 + 0.5 \times 0$	$\phi(0)$	1
0	1	$-0.5 \times 1 + 0.5 \times 0 + 0.5 \times 1$	$\phi(0)$	1
0	0	$-0.5 \times 1 + 0.5 \times 0 + 0.5 \times 0$	$\phi(-0.5)$	0

Example

We get the logical OR function from the choice:
$p = 2$, $w_0 = -0.5$, $w_1 = w_2 = 0.5$
and the treshold function

(picture here)

as activation function.

x_1	x_2	v	$\phi(v)$	$x_1 \vee x_2$
1	1	$-0.5 \times 1 + 0.5 \times 1 + 0.5 \times 1$	$\phi(0.5)$	1
1	0	$-0.5 \times 1 + 0.5 \times 1 + 0.5 \times 0$	$\phi(0)$	1
0	1	$-0.5 \times 1 + 0.5 \times 0 + 0.5 \times 1$	$\phi(0)$	1
0	0	$-0.5 \times 1 + 0.5 \times 0 + 0.5 \times 0$	$\phi(-0.5)$	0

Each unit/neuron performs a relatively simple job: receive input from neighbours or external sources and use this to compute an output signal which is propagated to other units.

Apart from this processing, a second task is the adjustment of the weights.

The system is inherently parallel in the sense that many units can carry out their computations at the same time.

During operation, units can be updated either

synchronously	all units update their activation simultaneously
asynchronously	each unit has a (usually fixed) probability of updating its activation at any time

Each unit/neuron performs a relatively simple job: receive input from neighbours or external sources and use this to compute an output signal which is propagated to other units.

Apart from this processing, a second task is the adjustment of the weights.

The system is inherently parallel in the sense that many units can carry out their computations at the same time.

During operation, units can be updated either

synchronously all units update their activation simultaneously

asynchronously each unit has a (usually fixed) probability of updating its activation at any time

Each unit/neuron performs a relatively simple job: receive input from neighbours or external sources and use this to compute an output signal which is propagated to other units.

Apart from this processing, a second task is the adjustment of the weights.

The system is inherently parallel in the sense that many units can carry out their computations at the same time.

During operation, units can be updated either

synchronously all units update their activation simultaneously

asynchronously each unit has a (usually fixed) probability of updating its activation at any time

Each unit/neuron performs a relatively simple job: receive input from neighbours or external sources and use this to compute an output signal which is propagated to other units.

Apart from this processing, a second task is the adjustment of the weights.

The system is inherently parallel in the sense that many units can carry out their computations at the same time.

During operation, units can be updated either

synchronously all units update their activation simultaneously

asynchronously each unit has a (usually fixed) probability of updating its activation at any time

There are mainly two flows of connections for a neural network:

Feed-forward here the data flow from input to output in a strictly feedforward manner.

The data processing can extend over multiple (layers of) units, but no feedback connections are present. That is, connections extending from outputs of units to inputs of units in the same layer or previous layers is not possible.

Recurrent here the data flow allows feedback connections.

In some cases, the activation values of the units undergo a relaxation process such that the neural network will evolve to a stable state.

There are mainly two flows of connections for a neural network:

Feed-forward here the data flow from input to output in a strictly feedforward manner.

The data processing can extend over multiple (layers of) units, but no feedback connections are present. That is, connections extending from outputs of units to inputs of units in the same layer or previous layers is not possible.

Recurrent here the data flow allows feedback connections.

In some cases, the activation values of the units undergo a relaxation process such that the neural network will evolve to a stable state.

There are mainly two flows of connections for a neural network:

Feed-forward here the data flow from input to output in a strictly feedforward manner.

The data processing can extend over multiple (layers of) units, but no feedback connections are present. That is, connections extending from outputs of units to inputs of units in the same layer or previous layers is not possible.

Recurrent here the data flow allows feedback connections.

In some cases, the activation values of the units undergo a relaxation process such that the neural network will evolve to a stable state.

Hebbian learning

Definition (Hebbian learning)

If neuron j repeatedly or persistently fires at neuron k (i.e. j fires 1 when output $k = 1$) then he strength of w_{kj} increases and vice versa

To describe a working algorithm we first introduce the notation

$$\begin{aligned} D &= \text{desired output} \\ A &= \text{actual output} \end{aligned}$$

where $A, D \in \{0, 1\}$ and then the learning rules

$$\begin{array}{ll} \text{If } D > A \text{ then} & w_{kj} := w_{kj} + x_j \\ \text{If } D < A \text{ then} & w_{kj} := w_{kj} - x_j \\ \text{If } D = A \text{ then} & \text{do nothing} \end{array}$$

Hebbian learning

Definition (Hebbian learning)

If neuron j repeatedly or persistently fires at neuron k (i.e. j fires 1 when output $k = 1$) then he strength of w_{kj} increases and vice versa

To describe a working algorithm we first introduce the notation

$$\begin{aligned} D &= \text{desired output} \\ A &= \text{actual output} \end{aligned}$$

where $A, D \in \{0, 1\}$ and then the learning rules

$$\begin{array}{ll} \text{If } D > A \text{ then} & w_{kj} := w_{kj} + x_j \\ \text{If } D < A \text{ then} & w_{kj} := w_{kj} - x_j \\ \text{If } D = A \text{ then} & \text{do nothing} \end{array}$$

Example OR

Example

Suppose we have a single neuron wich we want to learn the rules for $x_1 \vee x_2$

Solution

The logical table we want is

x_1	x_2	$D = x_1 \vee x_2$
1	1	1
1	0	1
0	1	1
0	0	0

and we start with the weights $w_0 = w_1 = w_2 = 0$ and the treshold function, ϕ, as activation function (i.e. $x_0 = 1$).

Example OR

Example

Suppose we have a single neuron wich we want to learn the rules for $x_1 \vee x_2$

Solution

The logical table we want is

x_1	x_2	$D = x_1 \vee x_2$
1	1	1
1	0	1
0	1	1
0	0	0

and we start with the weights $w_0 = w_1 = w_2 = 0$ and the treshold function, ϕ, as activation function (i.e. $x_0 = 1$).

Example OR (cont)

Solution

Rewrite the transfer function as

$$v = \sum_{j=0}^{2} w_j x_j = (w_0, w_1, w_2) \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

and we get the updating algorithm (think one column at the time)

$$\begin{pmatrix} w_0 & w_1 & w_2 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_0 & 1 & 1 & 1 & 1 \\ x_1 & 1 & 1 & 0 & 0 \\ x_2 & 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\phi : \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} A & 1 & 1 & 1 & \mathbf{1} \\ D & 1 & 1 & 1 & \mathbf{0} \end{pmatrix}$$

and since $D_4 < A_4$ we update the weights $(-)$.

Example OR (cont)

Solution

Rewrite the transfer function as

$$v = \sum_{j=0}^{2} w_j x_j = (w_0, w_1, w_2) \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

and we get the updating algorithm (think one column at the time)

$$\begin{pmatrix} w_0 & w_1 & w_2 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_0 & 1 & 1 & 1 & 1 \\ x_1 & 1 & 1 & 0 & 0 \\ x_2 & 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\phi : \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} A & 1 & 1 & 1 & \mathbf{1} \\ D & 1 & 1 & 1 & \mathbf{0} \end{pmatrix}$$

and since $D_4 < A_4$ we update the weights $(-)$.

Example OR (cont)

Solution

We get

$$\begin{aligned} w_0 &:= w_0 - x_0 = 0 - 1 = -1 \\ w_1 &:= w_1 - x_1 = 0 - 0 = 0 \\ w_2 &:= w_2 - x_2 = 0 - 0 = 0 \end{aligned}$$

and our new weights becomes $w_0 = -1$, $w_1 = 0$, $w_2 = 0$.

Next iteration

$$\underbrace{\begin{pmatrix} -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}}_{\begin{pmatrix} -1 & -1 & -1 & -1 \end{pmatrix}} \xrightarrow{\phi} \begin{pmatrix} A & \mathbf{0} & 0 & 0 & 0 \\ D & \mathbf{1} & 1 & 1 & 0 \end{pmatrix}$$

and since $A_1 < D_1$ *we update the weights* $(+)$.

Example OR (cont)

Solution

We get

$$\begin{aligned} w_0 &: = w_0 - x_0 = 0 - 1 = -1 \\ w_1 &: = w_1 - x_1 = 0 - 0 = 0 \\ w_2 &: = w_2 - x_2 = 0 - 0 = 0 \end{aligned}$$

and our new weights becomes $w_0 = -1$, $w_1 = 0$, $w_2 = 0$.
Next iteration

$$\underbrace{\begin{pmatrix} -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}}_{\begin{pmatrix} -1 & -1 & -1 & -1 \end{pmatrix}} \xrightarrow{\phi} \begin{pmatrix} A & \mathbf{0} & 0 & 0 & 0 \\ D & \mathbf{1} & 1 & 1 & 0 \end{pmatrix}$$

and since $A_1 < D_1$ *we update the weights* $(+)$.

Example OR (cont)

Solution

We get

$$\begin{aligned} w_0 &: = w_0 + x_0 = -1 + 1 = 0 \\ w_1 &: = w_1 + x_1 = 0 + 1 = 1 \\ w_2 &: = w_2 + x_2 = 0 + 1 = 1 \end{aligned}$$

and our new weights becomes $w_0 = 0$, $w_1 = 1$, $w_2 = 1$.

Next iteration

$$\underbrace{\begin{pmatrix} 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}}_{\begin{pmatrix} 2 & 1 & 1 & 0 \end{pmatrix}} \xrightarrow{\phi} \begin{pmatrix} A & 1 & 1 & 1 & \mathbf{1} \\ D & 1 & 1 & 1 & \mathbf{0} \end{pmatrix}$$

and since $D_4 < A_4$ *we update the weights* $(-)$.

Example OR (cont)

Solution

We get

$$\begin{aligned} w_0 &: = w_0 + x_0 = -1 + 1 = 0 \\ w_1 &: = w_1 + x_1 = 0 + 1 = 1 \\ w_2 &: = w_2 + x_2 = 0 + 1 = 1 \end{aligned}$$

and our new weights becomes $w_0 = 0$, $w_1 = 1$, $w_2 = 1$.
Next iteration

$$\underbrace{\begin{pmatrix} 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}}_{\begin{pmatrix} 2 & 1 & 1 & 0 \end{pmatrix}} \xrightarrow{\phi} \begin{pmatrix} A & 1 & 1 & 1 & \mathbf{1} \\ D & 1 & 1 & 1 & \mathbf{0} \end{pmatrix}$$

and since $D_4 < A_4$ *we update the weights* $(-)$.

Example OR (cont)

Solution

We get

$$\begin{aligned} w_0 &:= w_0 - x_0 = 0 - 1 = -1 \\ w_1 &:= w_1 - x_1 = 1 - 0 = 1 \\ w_2 &:= w_2 - x_2 = 1 - 0 = 1 \end{aligned}$$

and our new weights becomes $w_0 = -1$, $w_1 = 1$, $w_2 = 1$.

Next iteration

$$\underbrace{\begin{pmatrix} -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}}_{\begin{pmatrix} 1 & 0 & 0 & -1 \end{pmatrix}} \xrightarrow{\phi} \begin{pmatrix} A & 1 & 1 & 1 & 0 \\ D & 1 & 1 & 1 & 0 \end{pmatrix}$$

and since $D_i = A_i$ *for all* $i = 1, 2, 3, 4$ *the iteration stops.*

Example OR (cont)

Solution

We get

$$\begin{aligned} w_0 &: = w_0 - x_0 = 0 - 1 = -1 \\ w_1 &: = w_1 - x_1 = 1 - 0 = 1 \\ w_2 &: = w_2 - x_2 = 1 - 0 = 1 \end{aligned}$$

and our new weights becomes $w_0 = -1$, $w_1 = 1$, $w_2 = 1$.
Next iteration

$$\underbrace{\begin{pmatrix} -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}}_{\begin{pmatrix} 1 & 0 & 0 & -1 \end{pmatrix}} \xrightarrow{\phi} \begin{pmatrix} A & 1 & 1 & 1 & 0 \\ D & 1 & 1 & 1 & 0 \end{pmatrix}$$

and since $D_i = A_i$ *for all* $i = 1, 2, 3, 4$ *the iteration stops.*

Example OR (cont)

Solution

Hence the final weights becomes $w_0 = -1$, $w_1 = 1$, $w_2 = 1$

This example shows that it is possible to find weights that make a neuron equivalent to $x_1 \vee x_2$.

Example OR (cont)

Solution

Hence the final weights becomes $w_0 = -1$, $w_1 = 1$, $w_2 = 1$

This example shows that it is possible to find weights that make a neuron equivalent to $x_1 \vee x_2$.

Example AND

Problem

Show that there is a neuron that match $x_1 \wedge x_2$. The truth table for AND is

x_1	x_2	$x_1 \wedge x_2$
1	1	1
1	0	0
0	1	0
0	0	0

Example XOR

Example

Then an example which has no solution with **one** neuron. The logical table XOR

x_1	x_2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

can not be matched by one layer of neurons.

Example XOR (cont)

Proof.

If there is one solution then the we need simultaneous solution to the following four equations

$$\begin{aligned} w_0 + w_1 + w_2 &< 0 & (1) \\ w_0 + w_1 &\geq 0 & (2) \\ w_0 + w_2 &\geq 0 & (3) \\ w_0 &< 0 & (4) \end{aligned}$$

and this is not possible. Add equations 1 and 4 and also 2 and 3 and you get

$$\begin{aligned} 2w_0 + w_1 + w_2 &< 0 \\ 2w_0 + w_1 + w_2 &\geq 0 \end{aligned}$$

which obviously is impossible. □

Example XOR (cont)

Proof.

If there is one solution then the we need simultaneous solution to the following four equations

$$\begin{aligned} w_0 + w_1 + w_2 &< 0 \quad (1)\\ w_0 + w_1 &\geq 0 \quad (2)\\ w_0 + w_2 &\geq 0 \quad (3)\\ w_0 &< 0 \quad (4) \end{aligned}$$

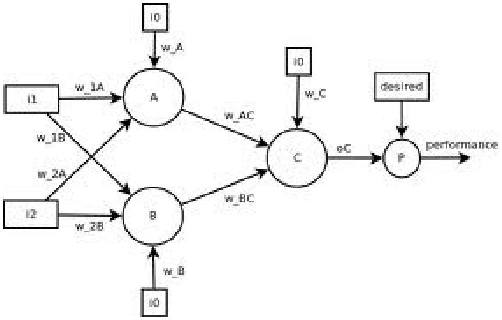
and this is not possible. Add equations 1 and 4 and also 2 and 3 and you get

$$\begin{aligned} 2w_0 + w_1 + w_2 &< 0\\ 2w_0 + w_1 + w_2 &\geq 0 \end{aligned}$$

which obviously is impossible. □

Example XOR (cont)

But if we introduce an extra layer of two neurons as in the picture below



then it is possible to get the table for XOR.

Example XOR (cont)

We can now construct neurons (that is: weights) such that

x_1	x_2	A_1
1	1	0
1	0	1
0	1	0
0	0	0

and

x_1	x_2	A_2
1	1	0
1	0	0
0	1	1
0	0	0

$\Rightarrow$

x_1	x_2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

Here A_1 and A_2 siginfies the first layer of neurons. It is left as an exercise to find the following sets of weights for each neuron

w_0	0.5	0.5	0.5
w_1	1	-1	1
w_2	-1	1	1

The layer A_1 and A_2, with two neurons, is called a hidden layer.

Example XOR (cont)

We can now construct neurons (that is: weights) such that

x_1	x_2	A_1
1	1	0
1	0	1
0	1	0
0	0	0

and

x_1	x_2	A_2
1	1	0
1	0	0
0	1	1
0	0	0

$\Rightarrow$

x_1	x_2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

Here A_1 and A_2 siginfies the first layer of neurons. It is left as an exercise to find the following sets of weights for each neuron

w_0	0.5	0.5	0.5
w_1	1	-1	1
w_2	-1	1	1

The layer A_1 and A_2, with two neurons, is called a hidden layer.

Example XOR (cont)

We can now construct neurons (that is: weights) such that

x_1	x_2	A_1		x_1	x_2	A_2		x_1	x_2	XOR
1	1	0		1	1	0		1	1	0
1	0	1	and	1	0	0	$\Rightarrow$	1	0	1
0	1	0		0	1	1		0	1	1
0	0	0		0	0	0		0	0	0

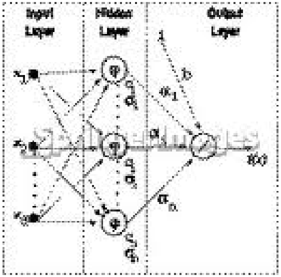
Here A_1 and A_2 siginfies the first layer of neurons. It is left as an exercise to find the following sets of weights for each neuron

w_0	0.5	0.5	0.5
w_1	1	-1	1
w_2	-1	1	1

The layer A_1 and A_2, with two neurons, is called a hidden layer.

1-hidden layer

The XOR table needed one hidden layer of neurons as in the, more general, picture below

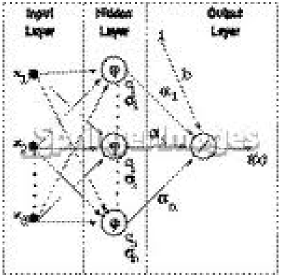


We will here give some results concerning the number of layers and the functions that may be produced with the neuron technique.

.

1-hidden layer

The XOR table needed one hidden layer of neurons as in the, more general, picture below

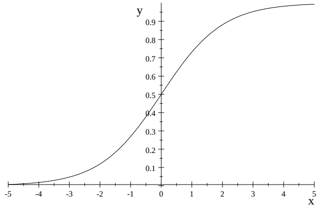


We will here give some results concerning the number of layers and the functions that may be produced with the neuron technique.

.

1-hidden layer

To do this we need another activation function, the sigmoid, $\phi(v) = \frac{1}{1+e^{-v}}$



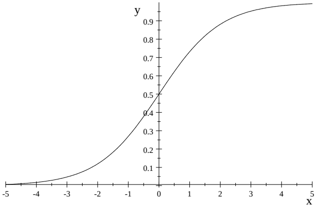
This function have the very nice property

$$\frac{d\phi(v)}{dv} = -\phi(v)(1-\phi(v))$$

which is heavily used in calculations for the more general case.

1-hidden layer

To do this we need another activation function, the sigmoid, $\phi(v) = \frac{1}{1+e^{-v}}$



This function have the very nice property

$$\frac{d\phi(v)}{dv} = -\phi(v)(1-\phi(v))$$

which is heavily used in calculations for the more general case.

The following theorem of Kolmogorov is crucial even if it is of more theoretical interest than practical.

Theorem (Kolmogorov 1957)

There exists fixed increasing continuous functions $h_{pq}(x)$ on $[0,1]$ so that each continuous function f on $[0,1]^n$ can be written in the form

$$f(x_1,\ldots,x_n)=\sum_{q=1}^{2n+1} g\left(\sum_{p=1}^{n} l_p h_q(x_p)\right)$$

where h_q are strictly monotonic functions of one variable and l_p are strictly positive constants smaller than 1.

Consequences of Kolmogorovs theorem

It has been shown that

1. Every boolean function can be represented exactly by a neural network with two layers.
2. Every bounded continuous function can be approximated by a neural network with two layers.
3. Any function can be approximated by a neural network with three layers.

In case 2) and 3) the activation function, for the hidden layers, must be the sigmoid and linear activation at the output.

Consequences of Kolmogorovs theorem

It has been shown that

1. Every boolean function can be represented exactly by a neural network with two layers.
2. Every bounded continuous function can be approximated by a neural network with two layers.
3. Any function can be approximated by a neural network with three layers.

In case 2) and 3) the activation function, for the hidden layers, must be the sigmoid and linear activation at the output.

Consequences of Kolmogorovs theorem

It has been shown that

1. Every boolean function can be represented exactly by a neural network with two layers.
2. Every bounded continuous function can be approximated by a neural network with two layers.
3. Any function can be approximated by a neural network with three layers.

In case 2) and 3) the activation function, for the hidden layers, must be the sigmoid and linear activation at the output.

Consequences of Kolmogorovs theorem

It has been shown that

1. Every boolean function can be represented exactly by a neural network with two layers.
2. Every bounded continuous function can be approximated by a neural network with two layers.
3. Any function can be approximated by a neural network with three layers.

In case 2) and 3) the activation function, for the hidden layers, must be the sigmoid and linear activation at the output.

Consequences of Kolmogorovs theorem

It has been shown that

1. Every boolean function can be represented exactly by a neural network with two layers.
2. Every bounded continuous function can be approximated by a neural network with two layers.
3. Any function can be approximated by a neural network with three layers.

In case 2) and 3) the activation function, for the hidden layers, must be the sigmoid and linear activation at the output.