Neural Networks in R

The neuralnet package

Fritz Günther

Colour Vision

Colour Vision

- Our `neuralnet` example will be on colour vision

Colour Vision

- Our `neuralnet` example will be on colour vision
- Since we want to use neural networks as psychological models, first some repetition on colour vision

Light

- The eyes' receptor cells react towards *light* produced by or reflected from objects

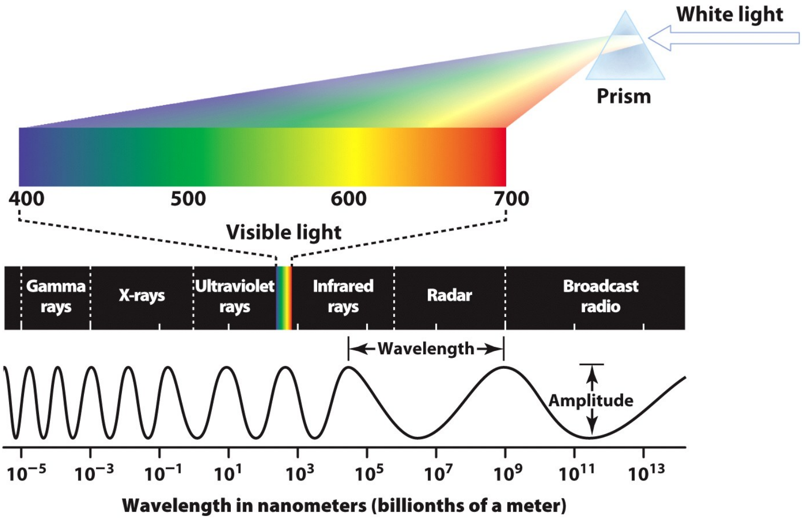
Light

- The eyes' receptor cells react towards *light* produced by or reflected from objects
- Light is (in part) an electromagnetic wave

Light

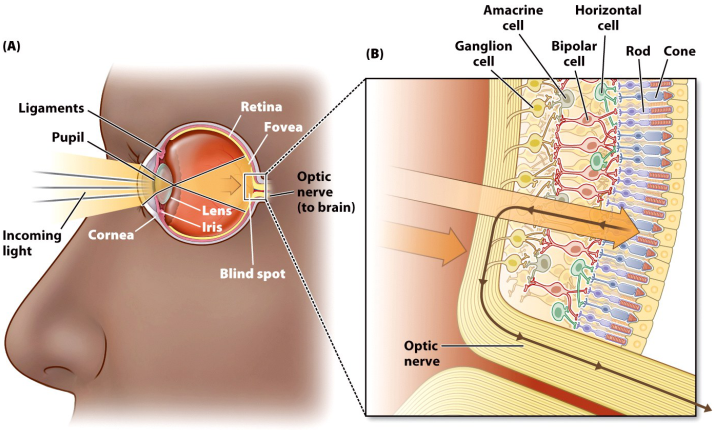
- The eyes' receptor cells react towards *light* produced by or reflected from objects
- Light is (in part) an electromagnetic wave
- *Visible spectrum:* For humans $\sim 360 - 750$ nm

Light



Psychology, 8/e Figure 4.23

The eye



Psychology, 8/e Figure 4.24

The eye

- Two types of photoreceptor cells

The eye

- Two types of photoreceptor cells
  - *rods*
  - *cones*

The eye

- Two types of photoreceptor cells
  - *rods*
  - *cones*
- Only cones enable the differentiation of chromatic light (= colour vision)

The eye

- Cones contain one of three different photopigments

The eye

- Cones contain one of three different photopigments
- These react with different intensity towards light of different wavelengths

Colour vision

- Humans can differentiate between millions of colours

Colour vision

- Humans can differentiate between millions of colours
- Three dimensions of colour:

- Humans can differentiate between millions of colours
- Three dimensions of colour:

  - Hue (wave length)

Colour vision

- Humans can differentiate between millions of colours
- Three dimensions of colour:

  - Hue (wave length)

  - Brightness

Colour vision

- Humans can differentiate between millions of colours
- Three dimensions of colour:

  - Hue (wave length)

  - Brightness

  - Saturation

Colour vision

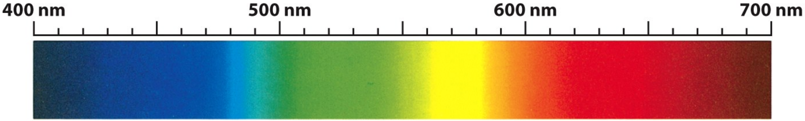
- Humans can differentiate between millions of colours
- Three dimensions of colour:

  - **Hue (wave length)**

  - Brightness

  - Saturation

Colour Vision



Psychology, 8/e Figure 4.32

Colour Vision

- *Young-Helmholtz-Theory*: *Tri-chromatic* colour vision, depending on three different types of cones (S, M, L)

Colour Vision

- *Young-Helmholtz-Theory*: *Tri-chromatic* colour vision, depending on three different types of cones (S, M, L)
- Cone types differ in the photopigments they contain

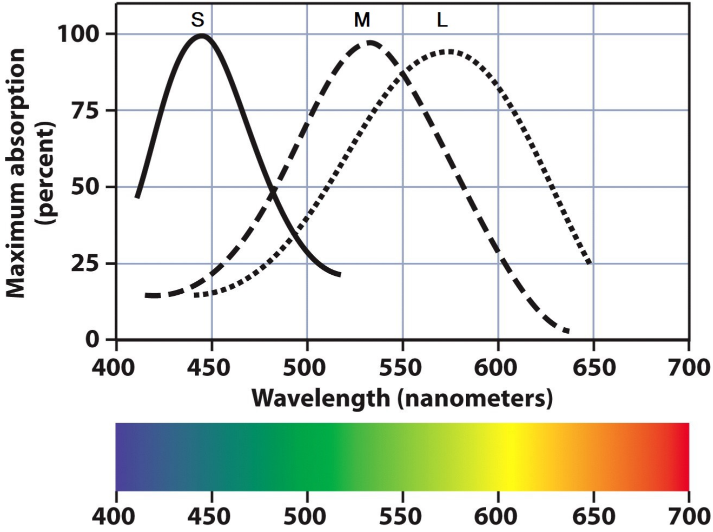
Colour Vision

- *Young-Helmholtz-Theory*: *Tri-chromatic* colour vision, depending on three different types of cones (S, M, L)
- Cone types differ in the photopigments they contain
- All cone types react, to some degree, towards all wave lengths

Colour Vision

- *Young-Helmholtz-Theory*: *Tri-chromatic* colour vision, depending on three different types of cones (S, M, L)
- Cone types differ in the photopigments they contain
- All cone types react, to some degree, towards all wave lengths
- Colour is therefore coded by the pattern of cone activities

Colour Vision



Psychology, 8/e Figure 4.34

Colour Vision

- What happens if light from different sources overlaps?

Colour Vision

- What happens if light from different sources overlaps?
- Receptor activity is the sum of activities for the different wave lengths

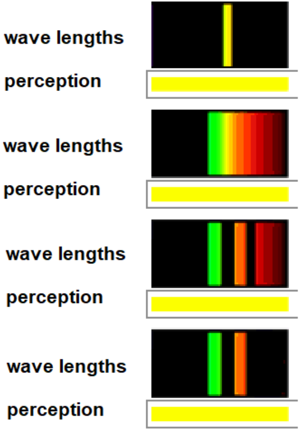
Colour Vision

- What happens if light from different sources overlaps?
- Receptor activity is the sum of activities for the different wave lengths
- If one receptor has a firing rate of 100, this may be caused by the following input:
  - Firing rate 100 for wave length A
  - Firing rate 10 for wave length A and 90 for wave length B
  - Firing rate 10 for A, 70 for B, 20 for C

Colour Vision

- What happens if light from different sources overlaps?
- Receptor activity is the sum of activities for the different wave lengths
- If one receptor has a firing rate of 100, this may be caused by the following input:
  - Firing rate 100 for wave length A
  - Firing rate 10 for wave length A and 90 for wave length B
  - Firing rate 10 for A, 70 for B, 20 for C
- Best example: RGB colours (TV, computer monitors)

Colour Vision



Source: www.handprint.com

- *Additive Colour Mixing:* Through overlap of chromatic light

- *Additive Colour Mixing:* Through overlap of chromatic light
- Chromatic light is characterized by a certain distribution of wave lengths

Colour Vision

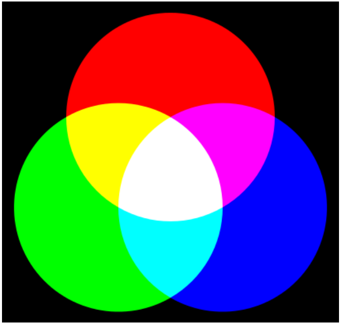
- *Additive Colour Mixing:* Through overlap of chromatic light
- Chromatic light is characterized by a certain distribution of wave lengths
- Through mixing, these distributions are *added*, resulting in a new distribution

Colour Vision

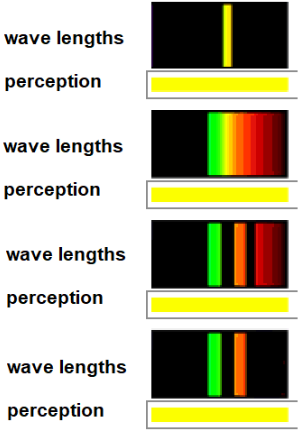
- *Additive Colour Mixing:* Through overlap of chromatic light
- Chromatic light is characterized by a certain distribution of wave lengths
- Through mixing, these distributions are *added*, resulting in a new distribution

  Achromatic light (white, black, greyscale): Uniform distribution

Colour Vision



Colour Vision



Source: www.handprint.com

- Colour vision is *not* a copy of the physical world

Colour Vision

- Additive Colour Mixing is a *physiological* (and ultimately *psychological*) phenomenon, based on our receptors and the processing of firing rates

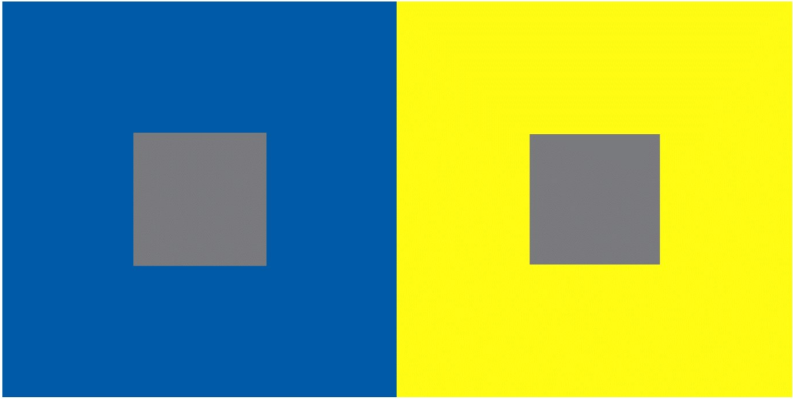
Colour Vision

- Additive Colour Mixing is a *physiological* (and ultimately *psychological*) phenomenon, based on our receptors and the processing of firing rates
- (In comparison, *substractive colour mixing* as done in painting is a physical phenomenon)

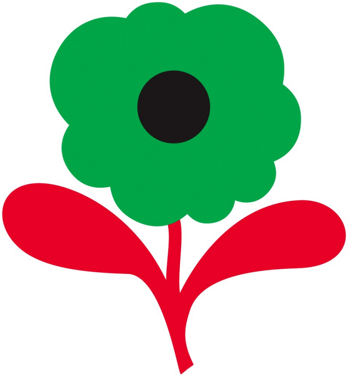
Colour Vision

- *Colour Contrast:* Areas adjacent to colours appear in their complementary colour

- *Colour Contrast:* Areas adjacent to colours appear in their complementary colour
- Complementary colours are blue - yellow; red - green (and black - white)



Psychology, 8/e Figure 4.35



Psychology, 8/e Figure 4.36

Colour Vision

Opponent-Process-Theory (Hering; Hurvich & Jameson)

- Theory postulates a layer of neurons in the visual system receiving input from the cones

Colour Vision

Opponent-Process-Theory (Hering; Hurvich & Jameson)

- Theory postulates a layer of neurons in the visual system receiving input from the cones
- These code the input in three pairs: red - green; blue - yellow; black - white

Colour Vision

Opponent-Process-Theory (Hering; Hurvich & Jameson)

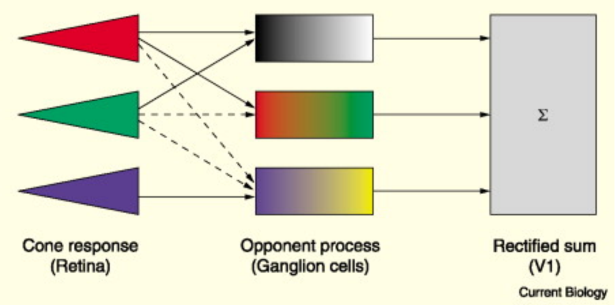
- Theory postulates a layer of neurons in the visual system receiving input from the cones
- These code the input in three pairs: red - green; blue - yellow; black - white
- Depending on the input, these neurons fire more (perception shifted towards one side of the pair) or less (perception shifts towards the other side)

Colour Vision

Opponent-Process-Theory (Hering; Hurvich & Jameson)

- Theory postulates a layer of neurons in the visual system receiving input from the cones
- These code the input in three pairs: red - green; blue - yellow; black - white
- Depending on the input, these neurons fire more (perception shifted towards one side of the pair) or less (perception shifts towards the other side)
- Example:
  Input shifts red-green to red and blue-yellow to blue
  $\Longrightarrow$ Perception purple
  Input shifts blue-yellow to blue but doesn't affect red-green
  $\Longrightarrow$ Perception blue

Colour Vision



Neural Networks - An overview

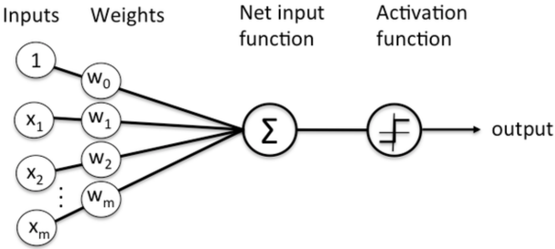
Neural Networks - Overview

Interviewer::	Why should we hire you?
Applicant:	I am an expert in machine learning.
Interviewer:	So you're good ad maths? What is $16 + 3$?
Applicant:	4
Interviewer:	That's not even close, it's 19!
Applicant:	13
Interviewer:	No, it's 19!
Applicant:	18
Interviewer:	No, 19!
Applicant:	19
Interviewer:	You're hired!

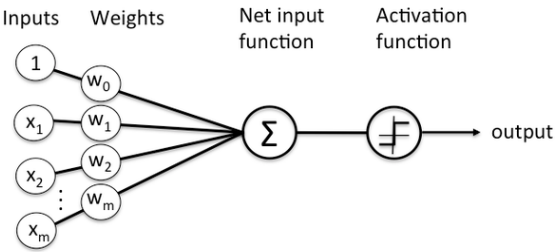
Neural Networks - Overview

- Nice overview about implementing neural networks in R can be found here:
  `https://selbydavid.com/2018/01/09/neural-network/`

- Neural Networks are similar to regression models: Predict outcomes from predictors
- They learn weights linking predictor values to outcome values



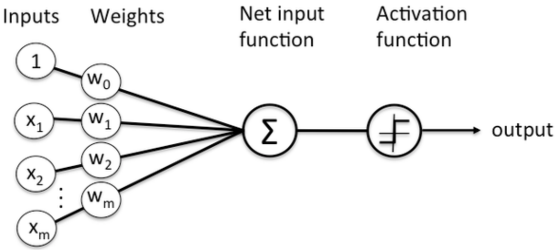
Neural Networks - Overview



- Computing the output (forward propagation):

$$y0 = \sum w_i \cdot x_i$$

Neural Networks - Overview

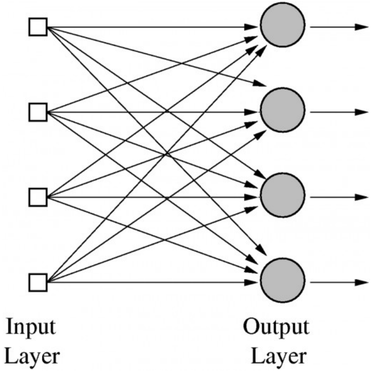


- Computing the output (forward propagation):

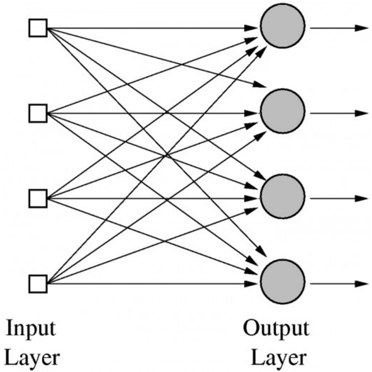
$$y = g(\sum w_i \cdot x_i)$$

, where g is the activation function

- Neural Networks can also predict multiple outcome values from a set of predictors



Neural Networks - Overview



- In that case, we have

$$y_j = g(\sum w_{ij} \cdot x_i)$$

Neural Networks - Overview

- Neural networks are typically *trained* to obtain the weights. Basic training procedure:
  - Start with random weights
  - Take a training item
  - Compute output from predictors (forward propagation)
  - Compute error between predicted output and actual output (supervised learning)
  - Adjust the weights according to the error (backpropagation)
  - Take the next training item and repeat these steps
  - Cycle through all training items until weights don't really change anymore

- Closely correspondes to learning in the Rescorla-Wagner model
- (1) Compute difference between predicted and actual output

$$E = \frac{1}{2}(t_j - y_j)^2$$

To compute a weight change value from the error, the derivative of the error function will enter the formula:

$$E' = (t_j - y_j)$$

- (2) Adjust (multiply) by *learning rate*

$$\alpha(t_j - y_j)$$

- (3) Change in weight linking input x_i to y_j is this product multiplied by input activation

$$\Delta w_{ij} = \alpha(t_j - y_j) \cdot x_i$$

- This is the core delta rule for linear activation functions

- (4) In the general case for any activation function, its derivation is applied to the weighted input and included

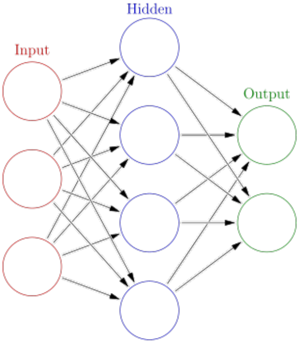
$$\Delta w_{ij} = \alpha(t_j - y_j)g'(h_j)x_i$$

With $h_j = \sum w_{ij}x_i$ and $y_j = g(h_j)$

- Training continues until the changes in weights Δw_{ij} no longer exceed a *threshold value* t. Every training cylce uses all training items.

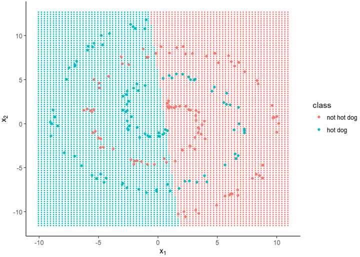
Neural Networks - Overview

- *Hidden layers* are intermediate levels between input and output
- Typically, they take input from all nodes in the previous layer, and give output to all nodes in the next layer
- In this case, it's easiest to consider them as multiple, chained neural networks where the output of layer n serves as input for layer $n+1$

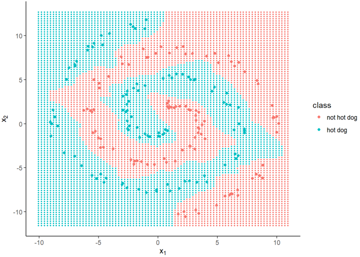


Neural Networks - Overview

- Hidden layers allow the network to deal with non-linearities
- Example: Predict color from x and y coordinates



without hidden layer



with hidden layer

Neural Networks - Overview

Play around with neural networks:

https://playground.tensorflow.org/

The `neuralnet` package

The `neuralnet` package

- Article describing the `neuralnet` package and its background:

  Günther, F., & Fritsch, S. (2010). neuralnet: Training of neural networks. *The R journal, 2(1),* 30-38.

  (The author is *Frauke* Günther, not me)

The `neuralnet` package

- Install the `neuralnet` package with
  `install.packages("neuralnet")`

The `neuralnet` package

- Install the `neuralnet` package with
  `install.packages("neuralnet")`
- Load the package with
  `library(neuralnet)`

The `neuralnet` package

- Load the `colors.txt` data set using
```
setwd("PATH_TO_DATA")
dat <- read.table("colors.txt")
```
  (or specify the path directly in the `read.table` command)

The `neuralnet` package

- The main function in the `neuralnet` package is the `neuralnet` function

The `neuralnet` package

- The main function in the `neuralnet` package is the `neuralnet` function
- This function trains a neural network from input data

The `neuralnet` package

- The main function in the `neuralnet` package is the `neuralnet` function
- This function trains a neural network from input data
- User defines network structure

The neuralnet package

- Usage:

```
neuralnet(formula, data, hidden = 1, threshold =
0.01, stepmax = 1e+05, rep = 1, startweights =
NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus =
1.2), learningrate=NULL, lifesign = "none",
lifesign.step = 1000, algorithm = "rprop+",
err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL,
constant.weights = NULL, likelihood = FALSE)
```

The neuralnet package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

`formula`	A formula specifying the input and output variables

As in all other models in R (such as `lm()` or `aov()`):

`out1 + out2 ~ var1 + var2 + var3`

The neuralnet package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

data	The data frame containing the input and output variables

The neuralnet package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

`hidden`	A vector specifying the hidden layer structure
`hidden=0`	No hidden layer
`hidden=c(4,5)`	Two hidden layers: First layer with 4 nodes, second layer with 5 nodes

The neuralnet package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

`threshold`	Specifies the threshold for weight adjustments (training is considered as converging if there are no more weight changes above the threshold)

The `neuralnet` package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

`stepmax`	Maximum number of training steps (One training step = one iteration over the whole data set)

The `neuralnet` package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

`rep`	Number of repetitions (i.e. how often the complete training algorithm is executed)

The neuralnet package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

`lifesign`	Observe training progress with `lifesign="full"`

The `neuralnet` package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

`algorithm` The learning algorithm (several included, see the `help`-function). Standard backpropagation `backprop` requires a `learningrate`

The neuralnet package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

`err.fct`	The error function, computing the difference between network-predicted and observed outcome. Sum of squared errors and cross-entropy are included, other (differentiable) functions can be provided

The neuralnet package

- Important Arguments:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01, stepmax =
1e+05, rep = 1, startweights = NULL, learningrate.limit = NULL,
learningrate.factor = list(minus = 0.5, plus = 1.2),
learningrate=NULL, lifesign = "none", lifesign.step = 1000,
algorithm = "rprop+", err.fct = "sse", act.fct = "logistic",
linear.output = TRUE, exclude = NULL, constant.weights = NULL,
likelihood = FALSE)
```

`act.fct`	Activation function computing the output value from the input values

The `neuralnet` package

Task: Train a single-layer (i.e., no hidden layers) network to predict the colour labels from the RGB code

Note: This is a physiological/psychological model, since additive colour mixing is not a physical phenomenon!

Inspecting the neural network

The `neuralnet` package

Inspecting the neural network

- Generic R functions
  `summary(network)`
  `str(network)`

The `neuralnet` package

An `nn` object contains the following elements (along with the input data):

The `neuralnet` package

An `nn` object contains the following elements (along with the input data):

`net.results`	The network's predicted output for the training data
`weights`	The trained network weights
`result.matrix`	Several indices summarizing the model (AIC, BIC, number of steps, reached threshold, error)

The `neuralnet` package

Inspecting the neural network

The `neuralnet` package

Inspecting the neural network

- The `plot.nn` function:
  `plot.nn(network)`

Predict output for given input

The `neuralnet` package

Predict output for given input

- The `predict` function:
  `predict(network,testset)`

The `neuralnet` package

Predict output for given input

- The `predict` function:
  `predict(network,testset)`
- The `testset` needs to have the same input variables as specified for the network!

The `neuralnet` package

What do we learn from our single-layer network?
Does it make sense?

The `neuralnet` package

Task: Train a network with one hidden layer (three nodes) to predict the colour labels from the RGB code

(Why?)

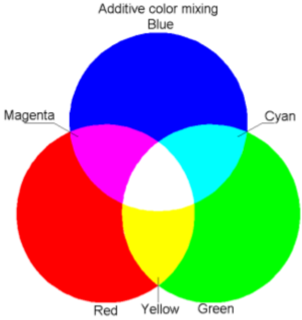
The `neuralnet` package

Is the hidden-layer network better than the single-layer network?

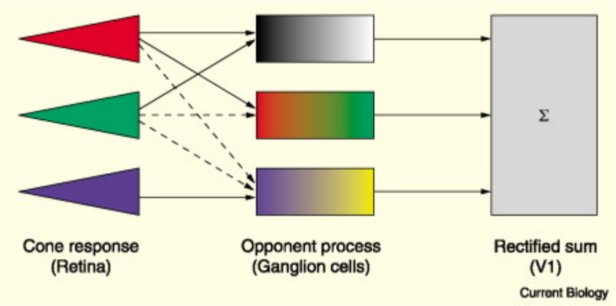
Does it work as expected?

In order to create the `colors.txt` data set, I just assigned colour labels on an intuitive basis

What happens when we apply a more “theory-conform” labelling system (including white and black)?



The `neuralnet` package



Does it help to include this *rectified sum* as an additional one-node hidden layer?