AIFH Vol 3: Deep Learning and Neural Networks

This document is a notebook of calculations for:

Artificial Intelligence for Humans Volume 3: Deep Learning and Neural Networks

For more information about the series visit: http://www.heatonresearch.com/aifh.

In the interest of open research, this document contains all of the R code used to create the figures for my book. Some of my calculations are also contained in this file. **This document is most not too useful by itself.** However, if you are wondering how I produced a chart or performed a calculation in my book, it is likely in this file.

–Jeff Heaton

Introduction

```
# Normalize weight
2000 - 100
```

```
## [1] 1900
```

```
5000 - 100
```

```
## [1] 4900
```

```
1900/4900
```

```
## [1] 0.3877551
```

```
# MNIST
28 * 28
```

```
## [1] 784
```

```
# Sunspots
3000 - 273.15
```

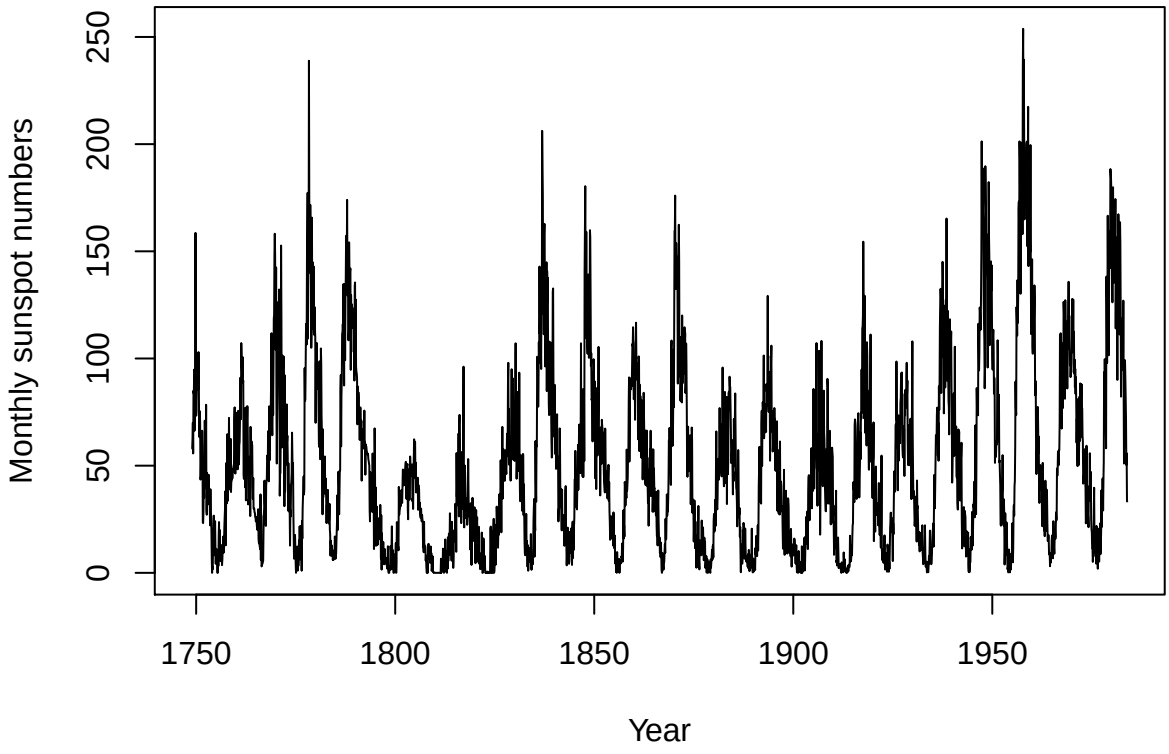
```
## [1] 2726.85
```

```
4500 - 273.15
```

```
## [1] 4226.85
```

Figure: Sunspots

```r
par(mar=c(5,4,2,2)+0.1)
require(graphics)
plot(sunspots, main = NULL, xlab = "Year",
     ylab = "Monthly sunspot numbers")
```



Chapter 1: Neural Network Basics

Equation 1.1: Neuron Output

$$f(x_i, w_i) = \phi(\sum_i (w_i \cdot x_i))$$

Equation 1.2: Linear Activation Function

$$\phi(x) = x$$

Equation 1.3: Step Activation Function

$$\phi(x) = \begin{cases} 1, & \text{if } x \geq 0.5. \\ 0, & \text{otherwise.} \end{cases}$$

Equation 1.4: Sigmoid Activation Function

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

Equation 1.5: Hyperbolic Tangent Activation Function

$$\phi(x) = \tanh(x)$$

Figure: Linear Activation Function

```
par(mar=c(5,4,2,2)+0.1)
act_linear <- function(x) x
plot(act_linear,xlim=c(-5,5),ylim=c(-5,5),xlab="x",ylab="y")
```

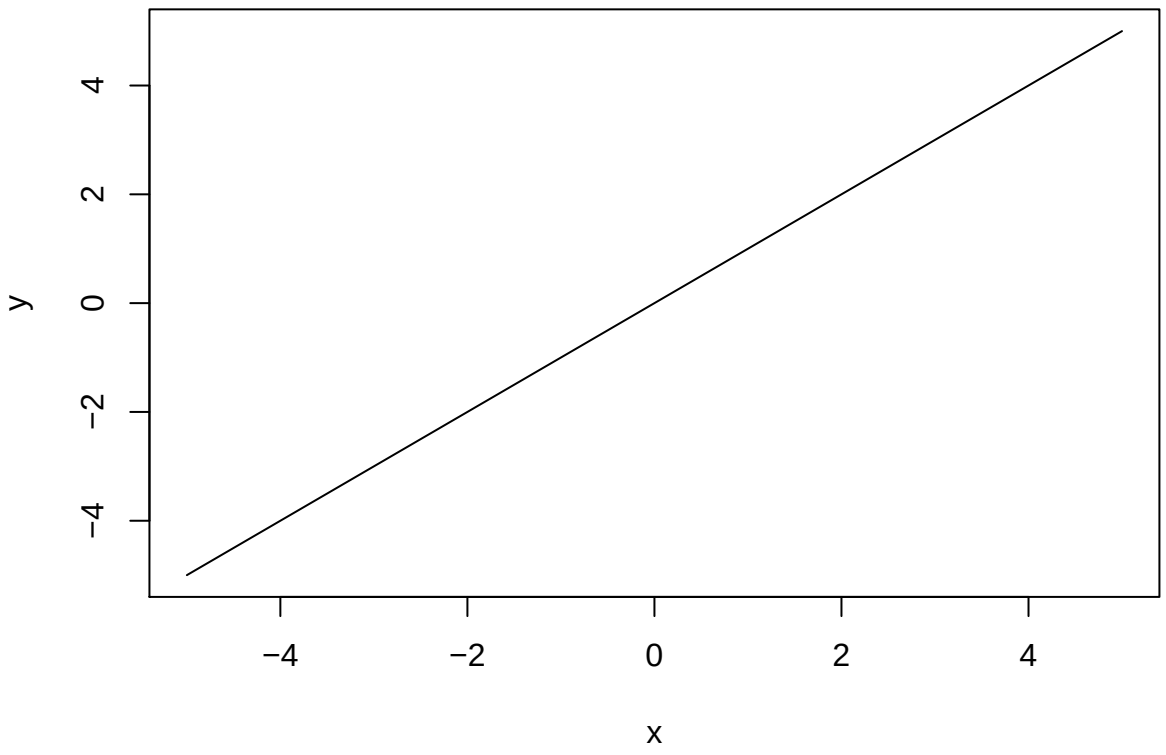


Figure: Step Activation Function

```
par(mar=c(5,4,2,2)+0.1)
act_step <- function(x) ifelse(x >= 0.5, 1, 0)
plot(act_step,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y")
```

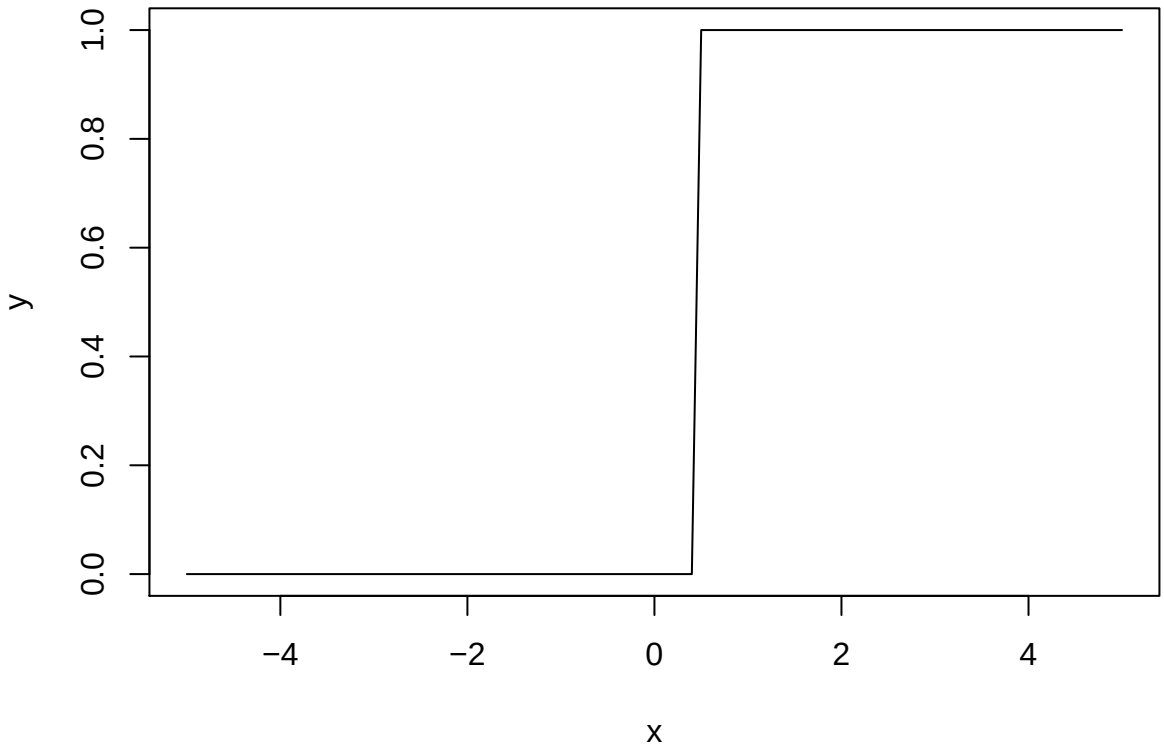


Figure: Sigmoid Activation Function

```
par(mar=c(5,4,2,2)+0.1)
act_sigmoid <- function(x) 1/(1+exp(-x))
plot(act_sigmoid,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y")
```

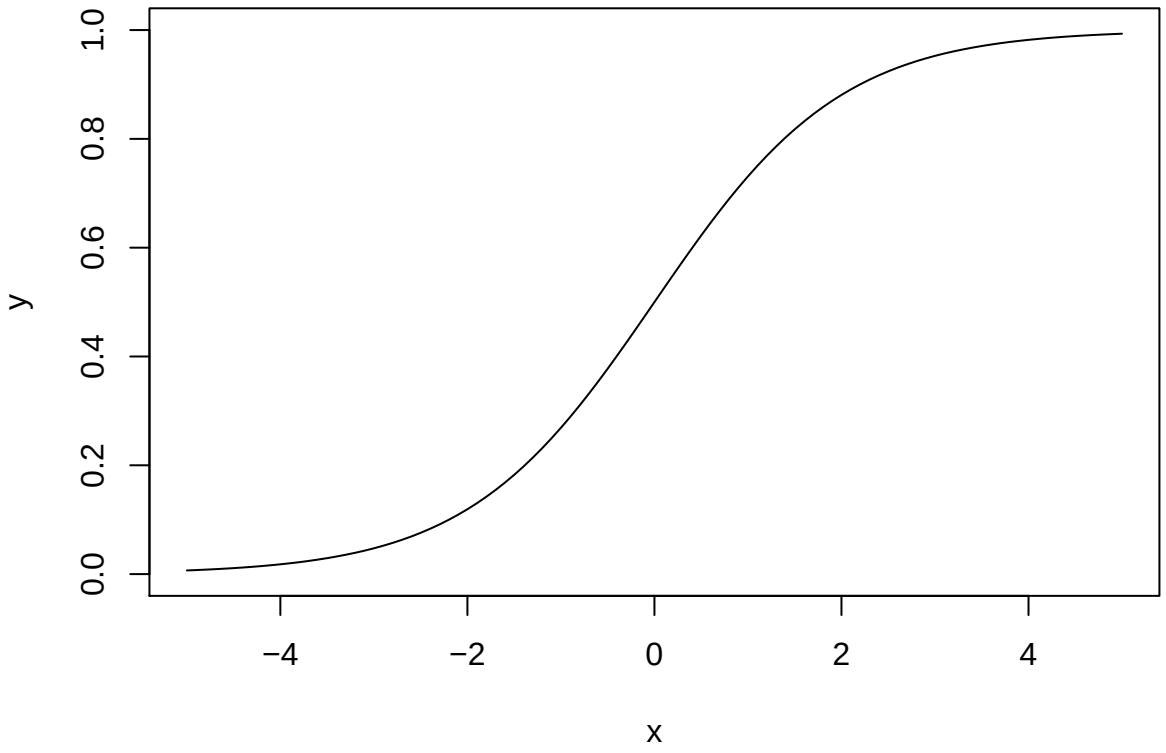


Figure: TanH Activation Function

```
par(mar=c(5,4,2,2)+0.1)
act_tanh <- function(x) tanh(x)
plot(act_tanh,xlim=c(-5,5),ylim=c(-1,1),xlab="x",ylab="y")
```

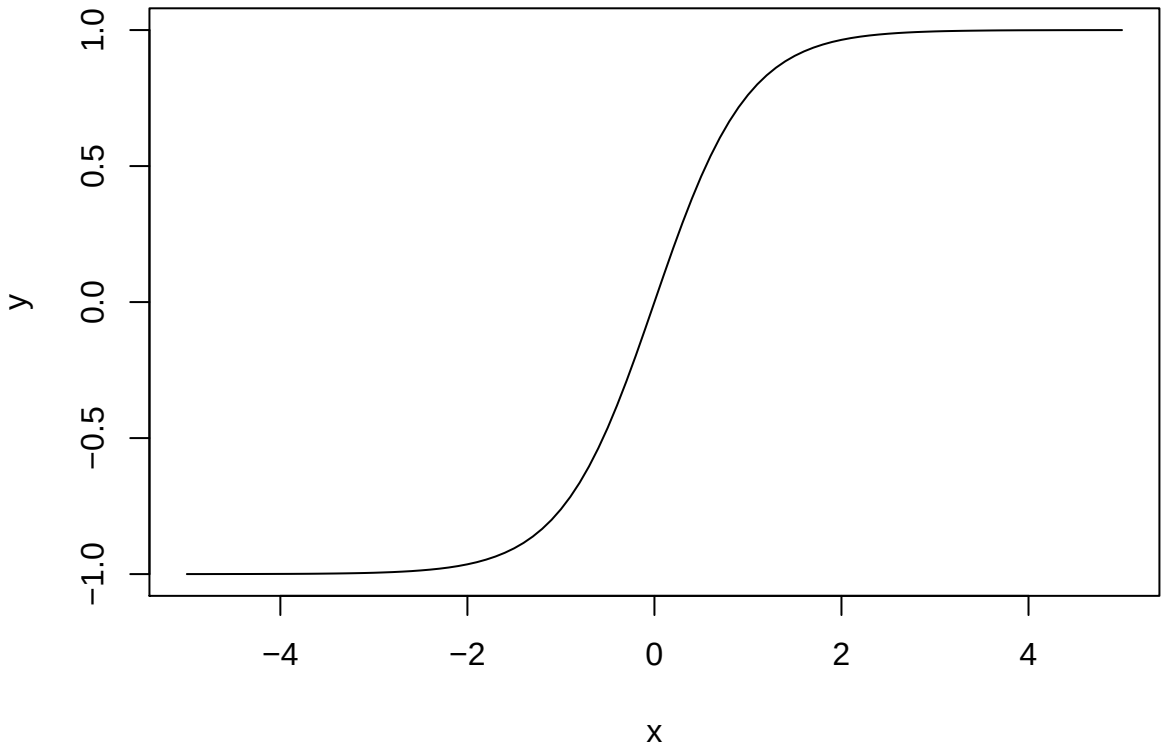
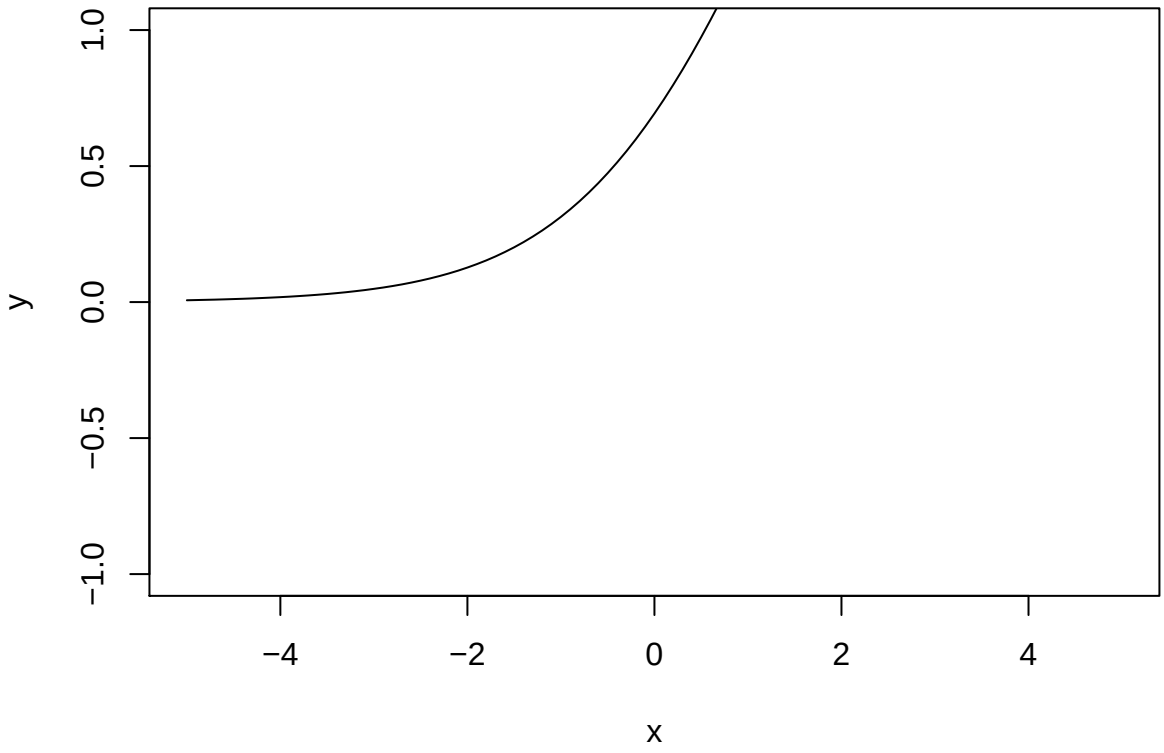


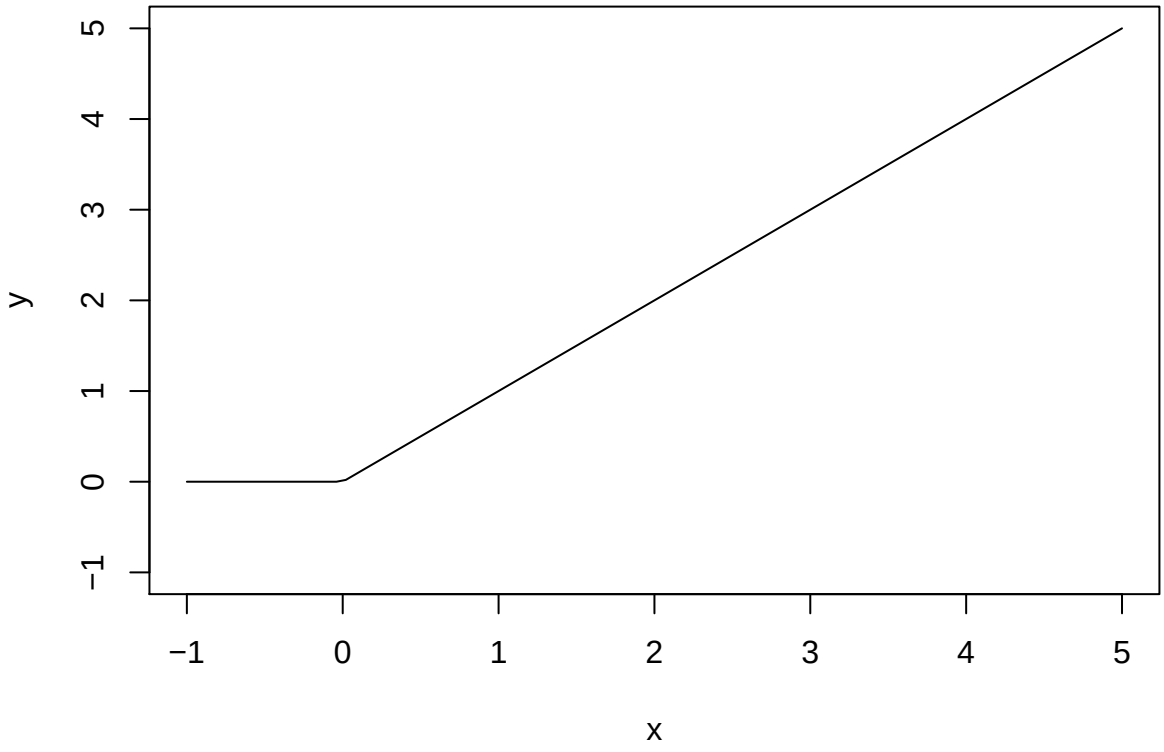
Figure: ReLU Activation Function

```
par(mar=c(5,4,2,2)+0.1)
act_tanh <- function(x) (log(1+exp(x)))
plot(act_tanh,xlim=c(-5,5),ylim=c(-1,1),xlab="x",ylab="y")
```



$$\phi(x) = \max(0, x)$$

```
par(mar=c(5,4,2,2)+0.1)
act_relu <- function(x) ifelse(x<0,0,x)
plot(act_relu,xlim=c(-1,5),ylim=c(-1,5),xlab="x",ylab="y")
```



Equation 5.2: Sigmoid Activation Function

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

Equation 1.7: The Softmax Function

$$\phi_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

Figure: Sigmoid Change Weight

```
par(mar=c(5,4,2,2)+0.1)
act_sigmoid <- function(x) 1/(1+exp(-x))
sig1 <- function(x) act_sigmoid(0.5*x)
sig2 <- function(x) act_sigmoid(1.0*x)
sig3 <- function(x) act_sigmoid(1.5*x)
sig4 <- function(x) act_sigmoid(2.0*x)
plot(act_sigmoid,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y")
plot(sig1,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y",add=TRUE)
plot(sig2,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y",add=TRUE)
plot(sig3,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y",add=TRUE)
plot(sig4,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y",add=TRUE)
```

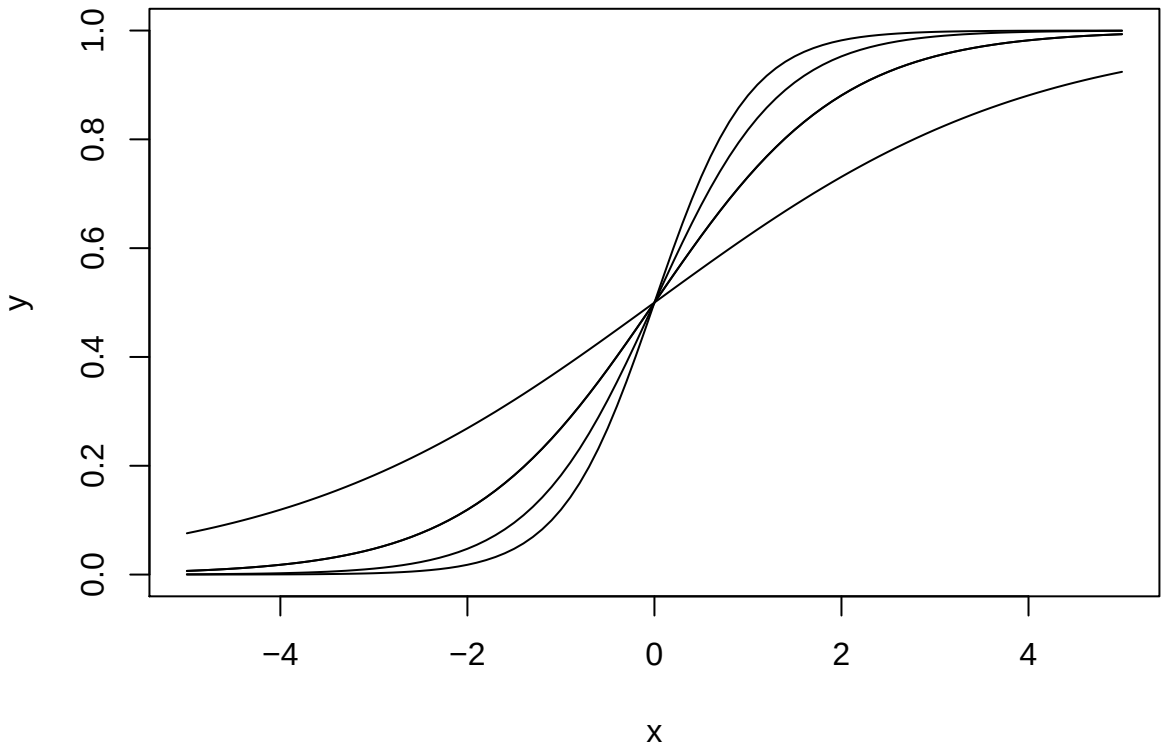
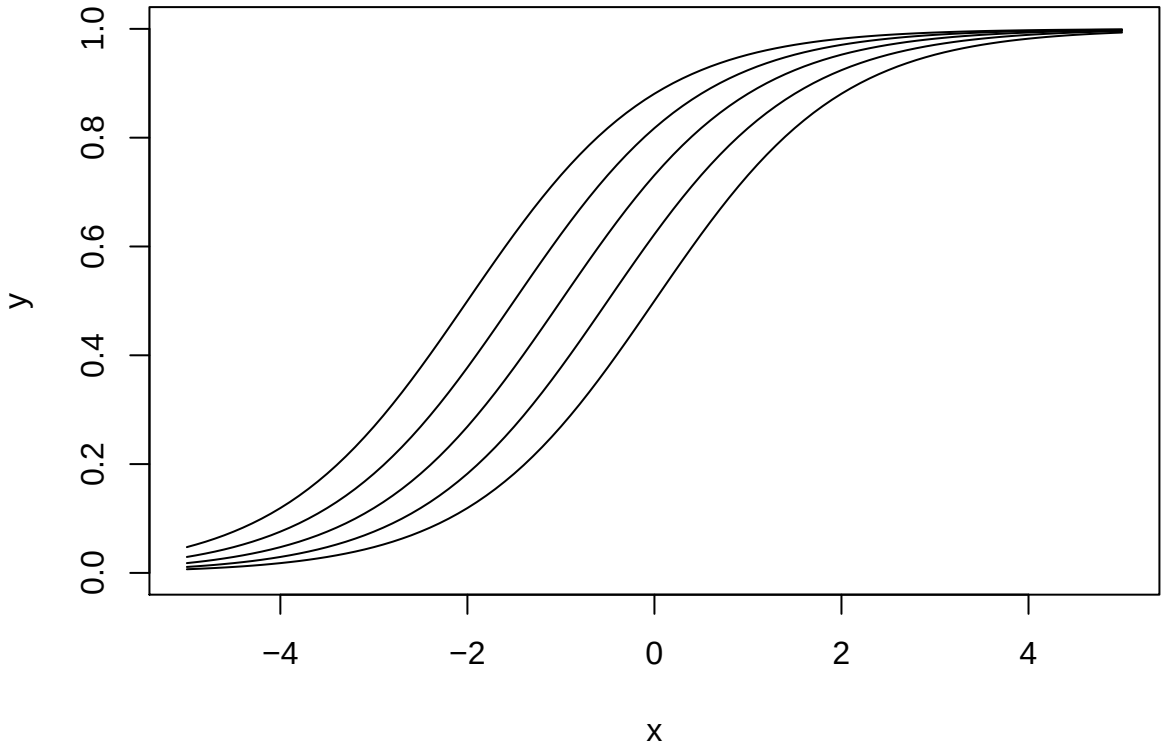


Figure: Sigmoid Change Bias

```
par(mar=c(5,4,2,2)+0.1)
act_sigmoid <- function(x) 1/(1+exp(-x))
sig1 <- function(x) act_sigmoid(1*x + 1*1)
sig2 <- function(x) act_sigmoid(1*x + 0.5*1)
sig3 <- function(x) act_sigmoid(1*x + 1.5*1)
sig4 <- function(x) act_sigmoid(1*x + 2*1)
plot(act_sigmoid,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y")
plot(sig1,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y",add=TRUE)
plot(sig2,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y",add=TRUE)
plot(sig3,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y",add=TRUE)
plot(sig4,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y",add=TRUE)
```



Logic Gates

```
act_step <- function(x) ifelse(x >= 0.5, 1, 0)
neuron1 <- function(x,w1,b1) act_step( (x*w1)+b1 )
neuron2 <- function(x1,x2,w1,w2,b1)  act_step((x1*w1) + (x2*w2) + b1)

# And
neuron2(0,0,1,1,-1.5)
```

```
## [1] 0
```

```
neuron2(1,0,1,1,-1.5)
```

```
## [1] 0
```

```
neuron2(0,1,1,1,-1.5)
```

```
## [1] 0
```

```
neuron2(1,1,1,1,-1.5)
```

```
## [1] 1
```

```
# Or
neuron2(0,0,1,1,-0.5)
```

```
## [1] 0
```

```
neuron2(1,0,1,1,-0.5)
```

```
## [1] 1
```

```
neuron2(0,1,1,1,-0.5)
```

```
## [1] 1
```

```
neuron2(1,1,1,1,-0.5)
```

```
## [1] 1
```

```
# Not
neuron1(0,-1,0.5)
```

```
## [1] 1
```

```
neuron1(1,-1,0.5)
```

```
## [1] 0
```

$$(NV \wedge LY) \vee (\neg LY \wedge PK)$$

$$p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$$

```
(0*1) + (1*1) - 0.5
```

```
## [1] 0.5
```

```
(0*1) + (1*1) - 1.5
```

```
## [1] -0.5
```

```
(0*-1)+0.5
```

```
## [1] 0.5
```

```
(1*1)+(1*1)-1.5
```

```
## [1] 0.5
```

Chapter 2: Self Organizing Maps

Equation 2.1: The Euclidean Distance between Weight and Output Neuron

$$\mathrm{d}(\mathbf{p}, \mathbf{w}) = \sqrt{\sum_{i=1}^{n}(p_i - w_i)^2}$$

Calculate the Euclidean norm of [2].

$$\|\mathbf{2}\| = \sqrt{2^2} = 2$$

Calculate the Euclidean norm of [2,3].

```
sqrt(2^2+3^2)
```

```
## [1] 3.605551
```

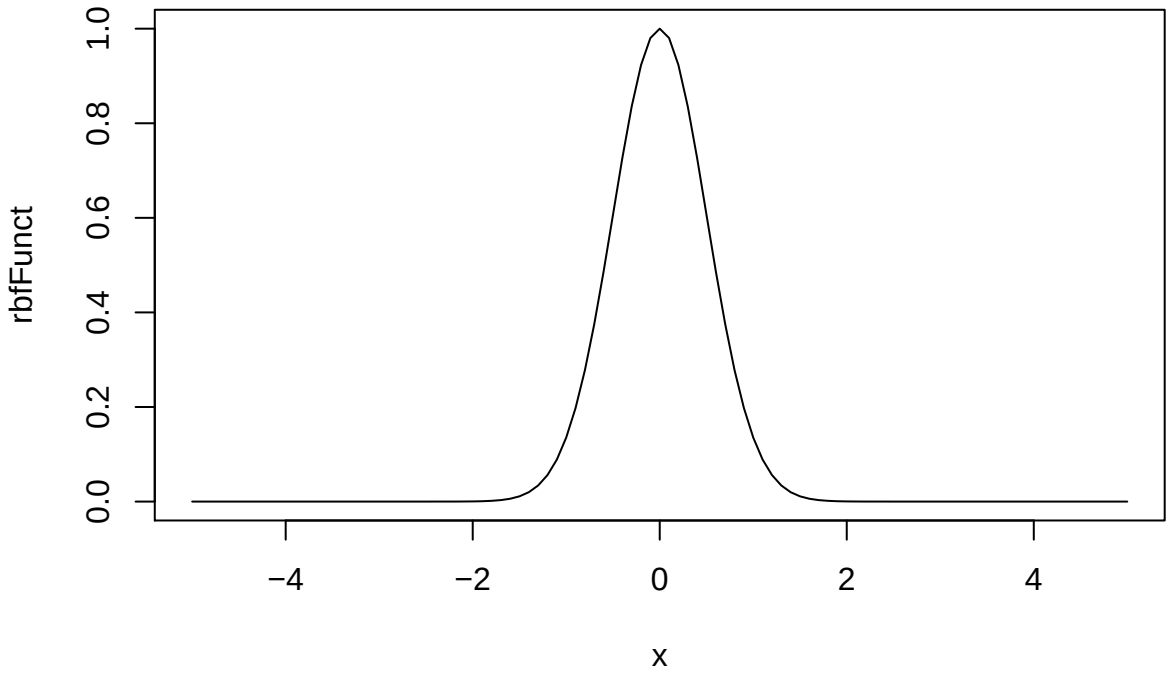
$$\|[\mathbf{2}, \mathbf{3}]\| = \sqrt{2^2 + 3^2} = 3.605551$$

Equation 2.2: SOM Learning Function

$$W_v(t+1) = W_v(t) + \theta(v,t)\alpha(t)(D(t) - W_v(t))$$

Figure: Gaussian Function

```
rbfFunct <- function(x) exp( 2*(-x^2) )
plot(rbfFunct,xlim=c(-5,5),ylim=c(0,1))
```



Equation 2.3: Gaussian Function

$$f(x, c, w) = e^{-(\frac{\|x-c\|)^2}{2w^2}}$$

Figure: Mexican Hat 2D

```
rbfFunct<-function(x)
{
  norm <- x*x
  (1 - norm) * exp(-norm / 2)
}
plot(rbfFunct,xlim=c(-5,5),ylim=c(-0.5,1))
```

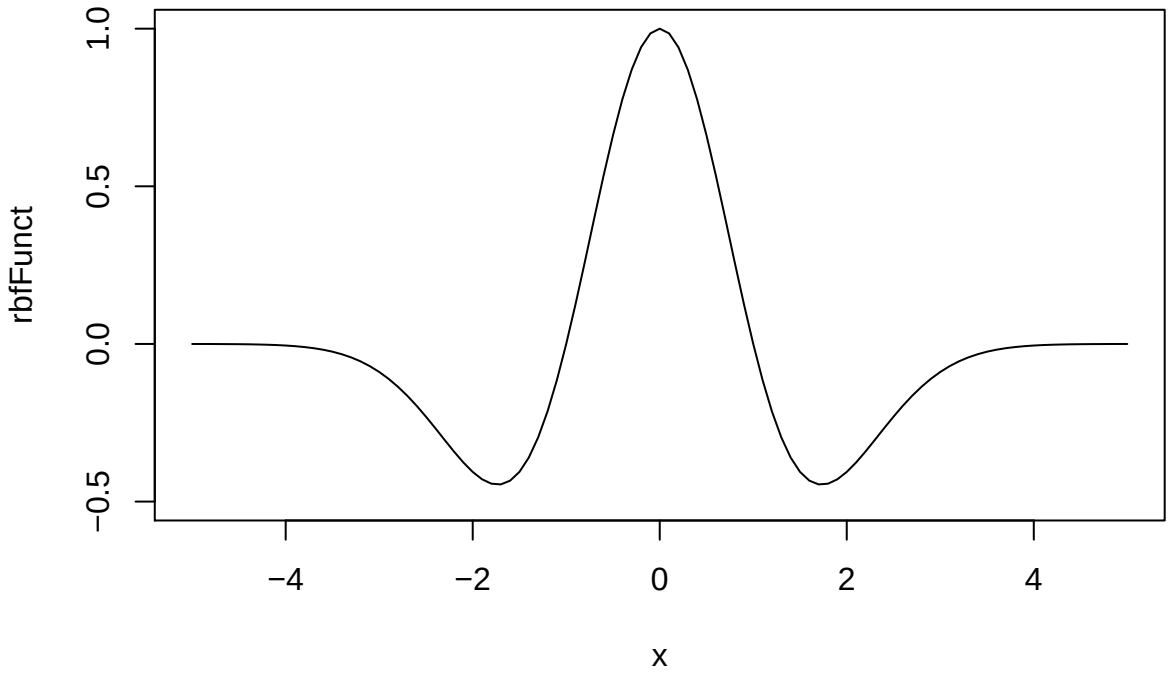
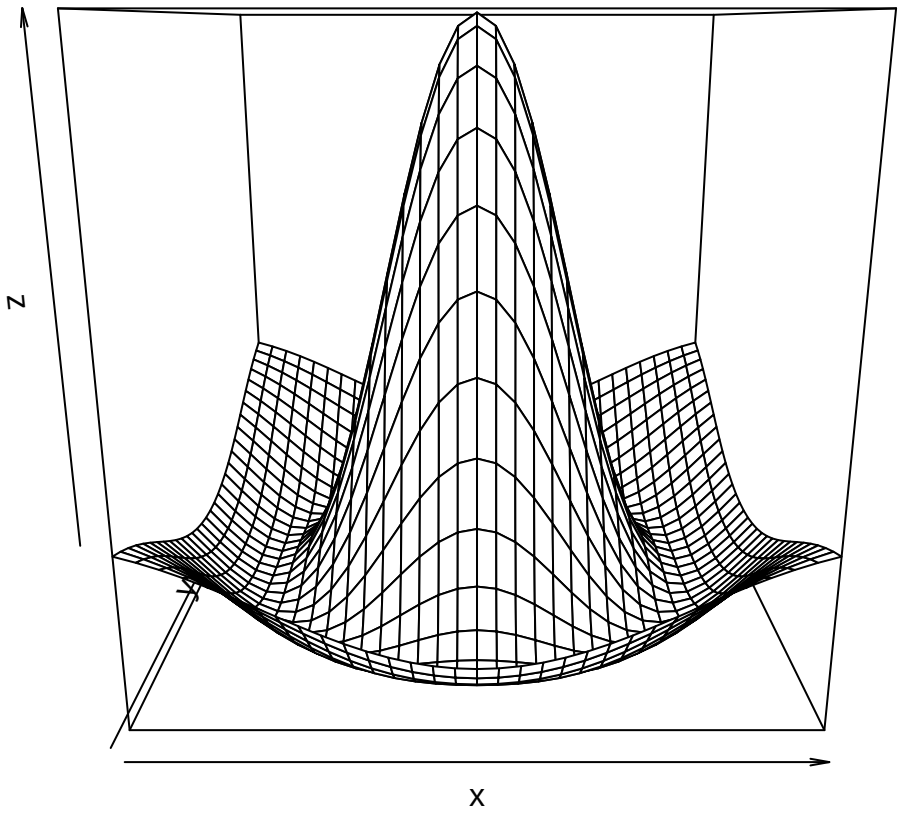


Figure: Mexican Hat Function 3D

```
par(mar=c(2,2,0,2)+0.1)
x<-seq(-4,4,0.25)
y<-seq(-4,4,0.25)
f<-function(x,y)
{
  w <- 3
  norm <- (x*x)+(y*y)
  (1 - (norm/w)) * exp(-norm / (2*w) )
}
z<-outer(x,y,f)
persp(x,y,z)
```



Equation: Mexican Hat

$$f(x,c,w) = \Big(1 - \frac{\|x-c\|^2}{w}\Big) e^{-\frac{\|x-c\|^2}{2w}}$$

Figure: Hexagon & Circle

```
library("plotrix")
par(mar=c(2,2,0,2)+0.1)
x = c( -1,      -0.5,        0.5,         1,       0.5,      -0.5, -1)
y = c( 0, -sqrt(0.75), -sqrt(0.75),         0, sqrt(0.75), sqrt(0.75),  0)
plot( x, y, type="l", asp = 1, xlim = c(-1, 1), ylim = c(-1, 1))
draw.circle(0, 0, 1, nv = 1000, border = NULL, col = NA, lty = 1, lwd = 1)
segments(0,0,-1,0)
segments(0,0,-0.5,-sqrt(0.75))
segments(0,0, 0.5,-sqrt(0.75))
segments(0,0, 1,0)
segments(0,0, 0.5,sqrt(0.75))
segments(0,0, -0.5,sqrt(0.75))
text(-0.5,0.1,"r")
text(-0.7,0.4,"s")
```

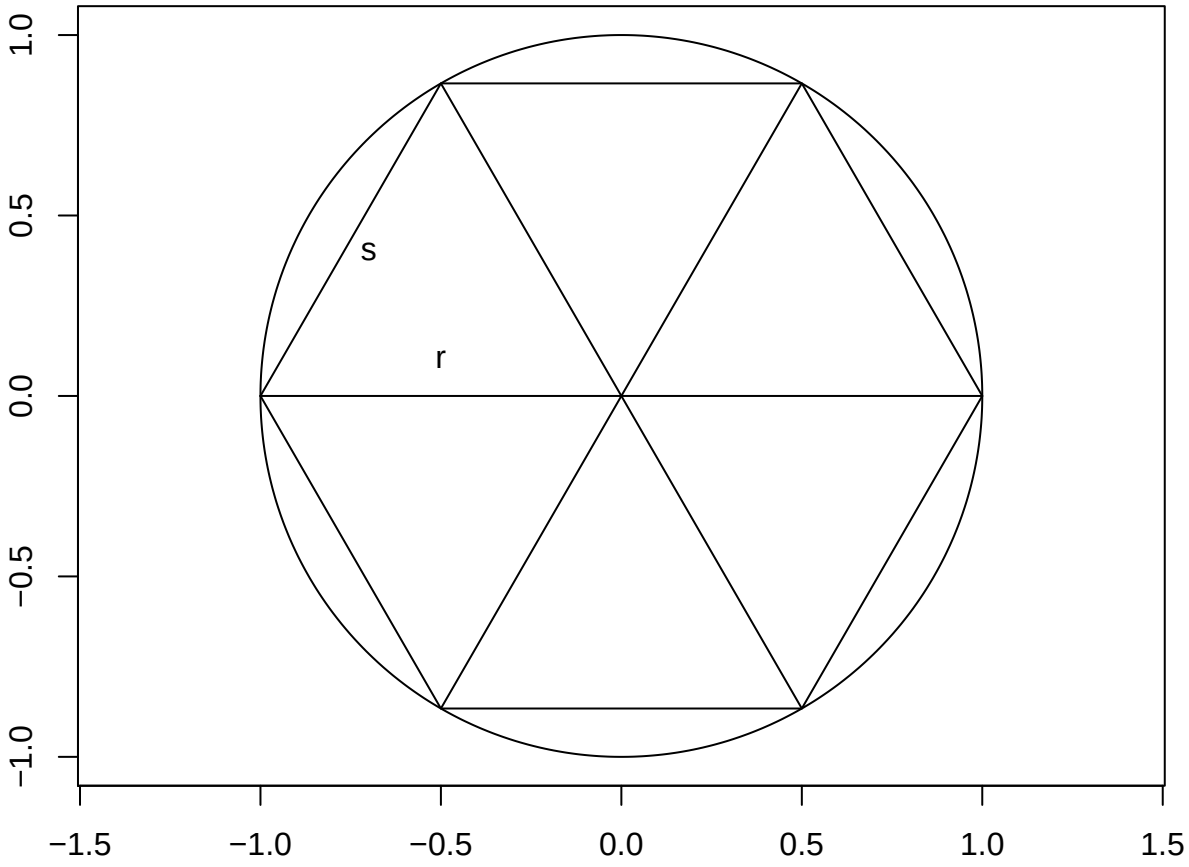
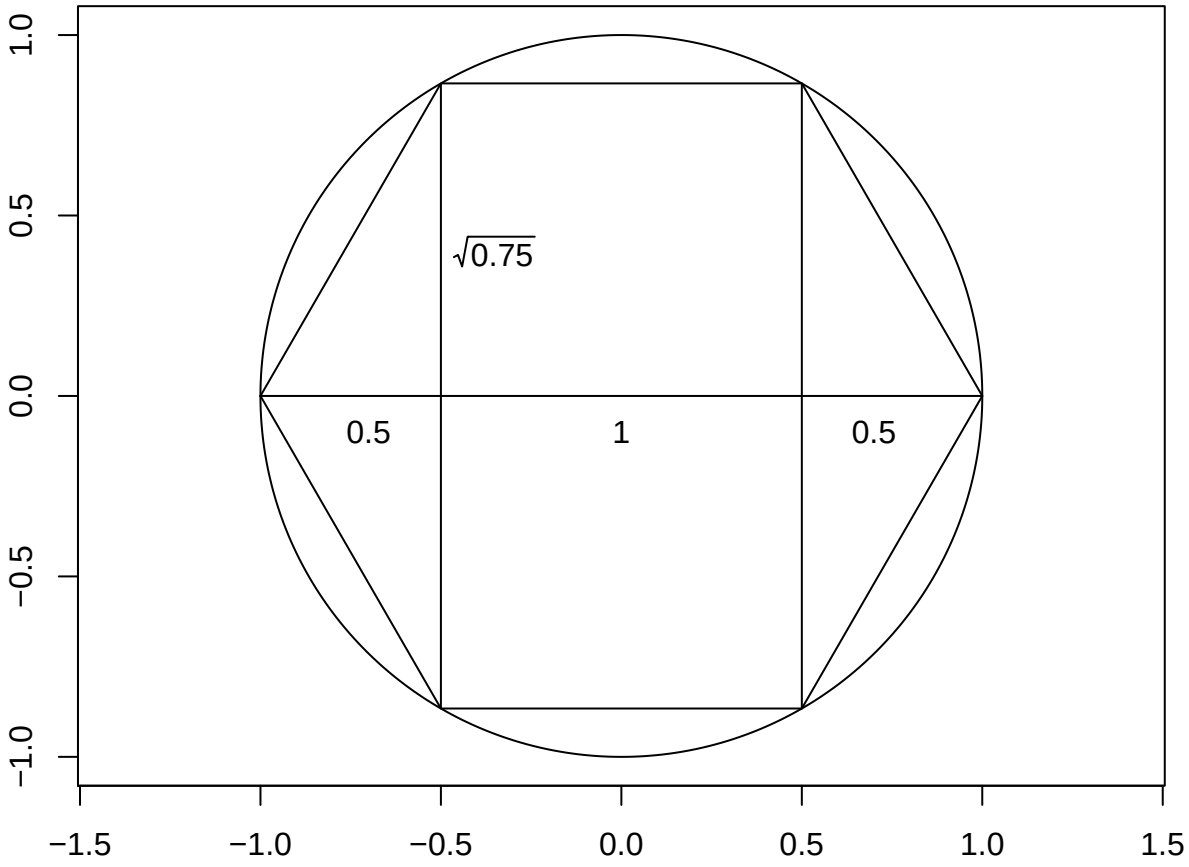


Figure: Hexagon Units

```
par(mar=c(2,2,0,2)+0.1)
x = c( -1,      -0.5,        0.5,         1,       0.5,      -0.5, -1)
y = c( 0, -sqrt(0.75), -sqrt(0.75),         0, sqrt(0.75), sqrt(0.75),  0)
plot( x, y, type="l", asp = 1, xlim = c(-1, 1), ylim = c(-1, 1))
draw.circle(0, 0, 1, nv = 1000, border = NULL, col = NA, lty = 1, lwd = 1)
segments(-1,0, 1,0)
segments(-0.5,-sqrt(0.75), -0.5, sqrt(0.75))
segments(0.5,-sqrt(0.75),0.5,sqrt(0.75))
text(-0.35, 0.4,expression( sqrt(0.75)  ) )
text(-0.7, -0.1,expression( 0.5  ) )
text(0.7, -0.1,expression( 0.5  ) )
text(0,-0.1,"1")
```



Chapter 3: Hopfield & Boltzmann Machines

Hopfield Update

$$s_i \leftarrow \begin{cases} +1 & \text{if } \sum_j w_{ij} s_j \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases}$$

Hopfield Hebbian Learning

$$w_{ij} = \frac{1}{n} \sum_{\mu=1}^{n} \epsilon_i^\mu \epsilon_j^\mu$$

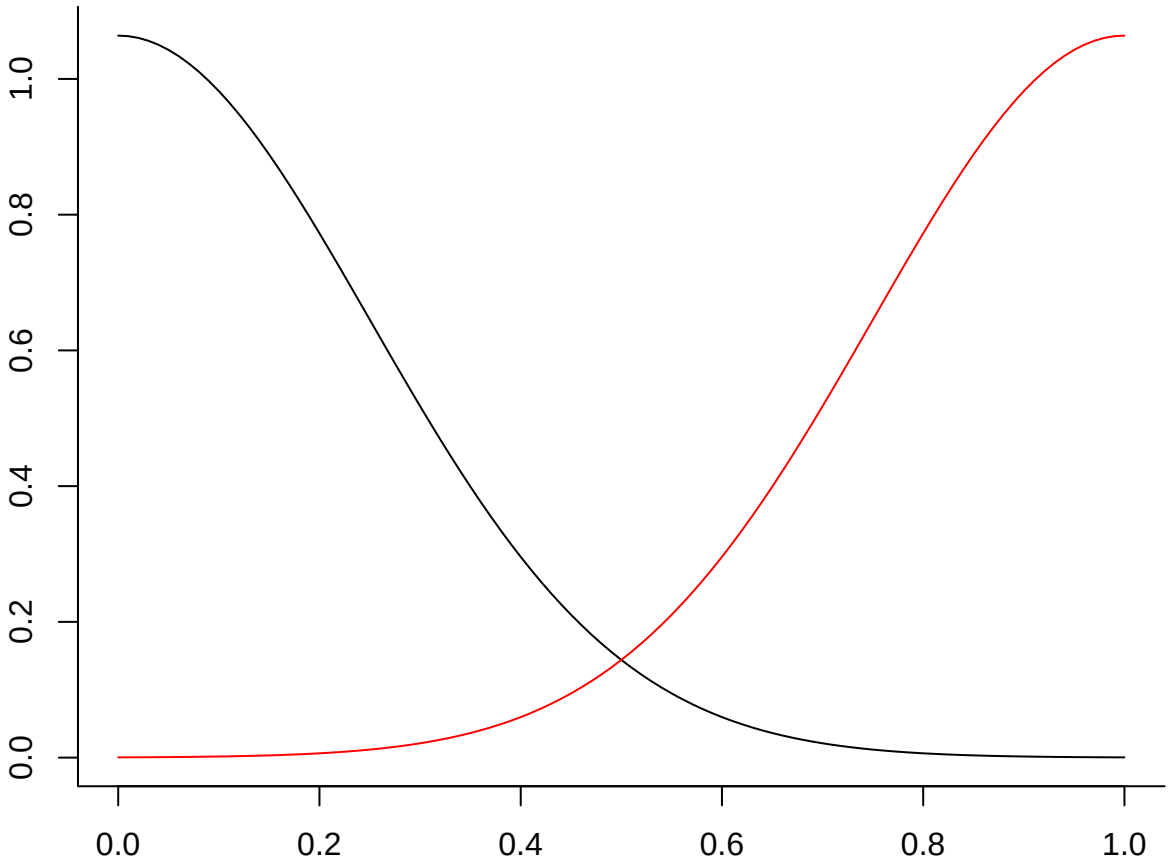
Hopfield Storkey Local Field

$$h_{ij} = \sum_{k=1, k\neq i,j} w_{ik} \epsilon_k$$

Hopfield Storkey Learning

$$\Delta w_{ij} = \frac{1}{n} \epsilon_i \epsilon_j - \frac{1}{n} \epsilon_i h_{ji} - \frac{1}{n} \epsilon_j h_{ij}$$

```
par(mar=c(2,2,0,2)+0.1)
x <- seq(0,1,0.01)
y1 <- dnorm(x,0,0.25)/1.5
y2 <- dnorm(x,1,0.25)/1.5
plot(x,y1,type="l",bty="L",ylab="Output",xlab="P(False)")
points(x,y2,type="l",col="red")
```




```
par(mar=c(2,2,0,2)+0.1)
x <- seq(0,1,0.01)
y1 <- dnorm(x,0,0.25)/1.5
y2 <- dnorm(x,1,0.25)/1.5
plot(x,y1,type="l",bty="L",ylab="Output",xlab="P(False)")
points(x,y2,type="l")
```

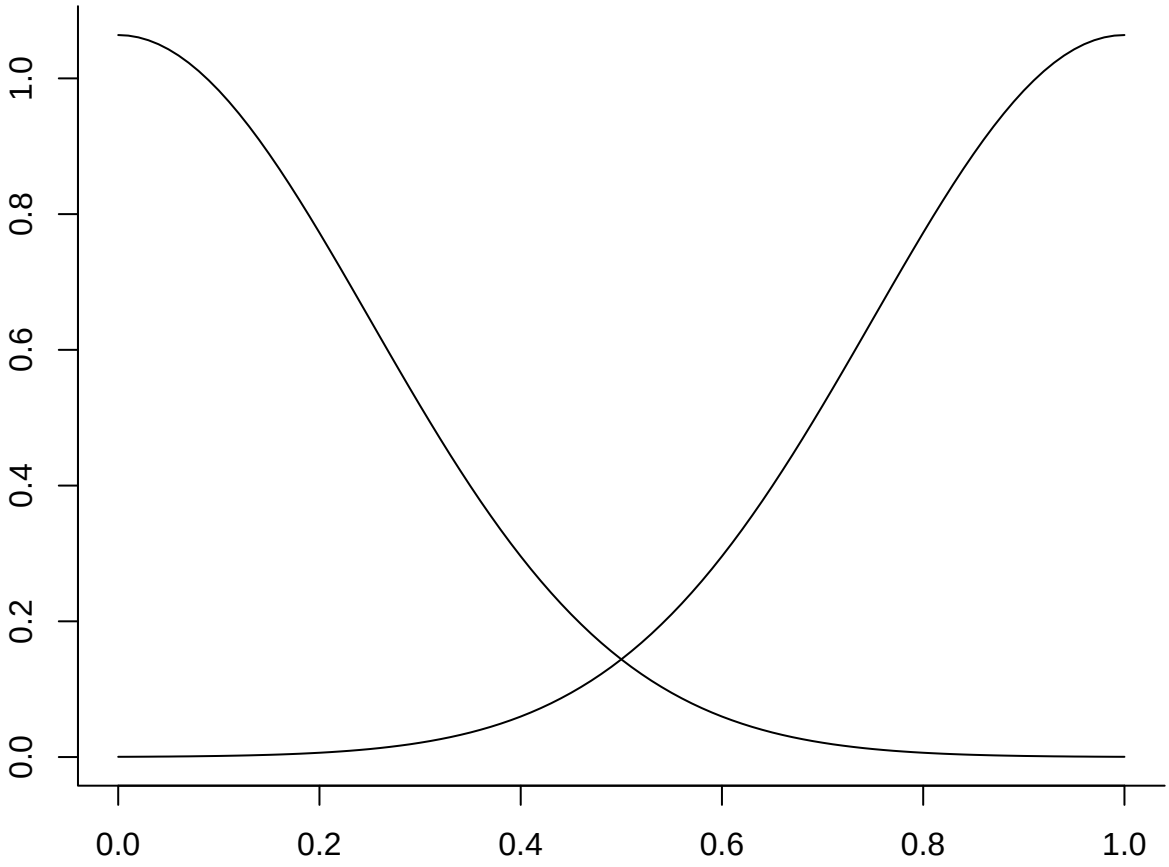
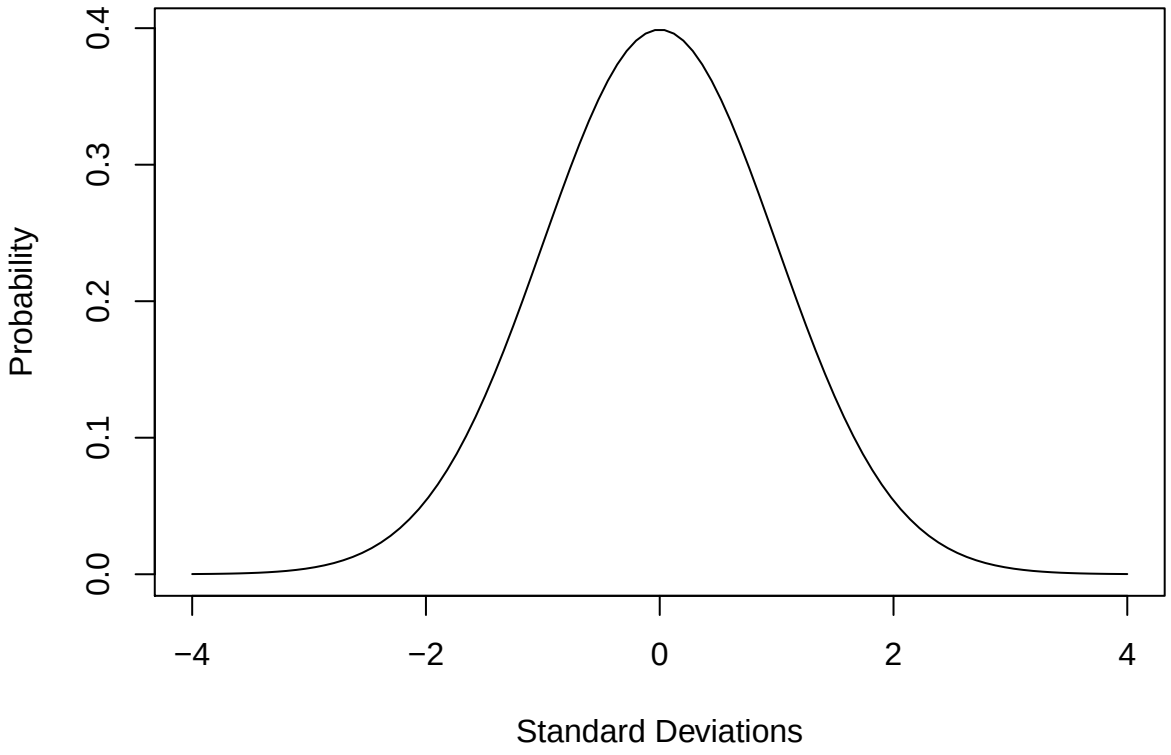


Figure: Normal Distrivution

```
par(mar=c(5,4,2,2)+0.1)
x <- seq(-4, 4, length=100)
hx <- dnorm(x)

plot(x, hx, type="l", xlab="Standard Deviations",
  ylab="Probability")
```



Equation 4.1: Standard Deviation for Xavier Algorithm

$$Var(W) = \frac{2}{n_{in} + n_{out}}$$

Normalize

$$norm(x, d_L, d_H, n_L, n_H) = \frac{(x - d_L)(n_H - n_L)}{(d_H - d_L)} + n_L$$

Denormalize

$$denorm(x, d_L, d_H, n_L, n_H) = \frac{(d_L - d_H)x - (n_H \cdot d_L) + d_H \cdot n_L}{(n_L - n_H)}$$

Mean

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Standard Deviation

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2}$$

Chapter 5: Training & Evaluation

Equation 5.5: Mean Square Error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

Chapter 6: Backpropagation Training

Equation 6.1: Mean Square Error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

Equation 6.2: Node Delta of MSE Output Layer

$$\delta_i = (\hat{y}_i - y_i)\phi_i'$$

Equation 6.3: Cross Entropy Error

$$CE = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)]$$

Equation 6.4: Node Delta of Cross Entropy Output Layer

$$\delta_i = \hat{y}_i - y_i$$

Equation 6.5: Calculating Interior Node Deltas

$$\delta_i = f_i' \sum_k w_{ki}\delta_k$$

Misc Equation: Node Deltas for Quadratic (not used in book)

$$\delta_i = \begin{cases} -Ef_i' & \text{, output nodes} \\ f_i' \sum_k w_{ki}\delta_k & \text{, interier nodes} \end{cases}$$

Equation 6.6: Derivative of the Linear Activation Function

$$\phi(x)\prime = 1$$

Equation 6.7: Softmax Activation Functon

$$\phi_i = \frac{e^{z_i}}{\sum\limits_{j \in group} e^{z_j}}$$

Equation 6.8: Derivative of the Softmax Activation Function

$$\frac{\partial \phi_i}{\partial z_i} = \phi_i(1 - \phi_i)$$

Equation 6.9: Derivative of the Sigmoid Activation Function

$$\phi(x)\prime = \phi(x)(1 - \phi(x))$$

Equation 6.10: Derivative of the Hyperbolic Tangent Activation Function

$$\phi(x)\prime = 1.0 - \phi(x)^2$$

Equation 6.11: Derivative of the ReLU Activation Function

$$\frac{dy}{dx}\phi(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Equation 6.12: Backpropagation Weight Update

$$\Delta w_{(t)} = \epsilon \frac{\partial E}{\partial w_{(t)}} + \alpha \Delta w_{(t-1)}$$

Figure 6.3: Tanh Activation Function & Derivative

```
par(mar=c(5,4,2,2)+0.1)
act_tanh <- function(x) tanh(x)
deriv_tanh <- function(x) 1.0 - act_tanh(x)^2
plot(act_tanh,xlim=c(-5,5),ylim=c(-1,1),xlab="x",ylab="y")
plot(deriv_tanh,xlim=c(-5,5),ylim=c(-1,1),xlab="x",ylab="y",add=TRUE, lty=2, col="red")
legend("bottomright", inset=.05,
  c('Tanh','Deriv. Tanh'), lwd=2, lty=c(1, 2), col=c("black","red"))
```

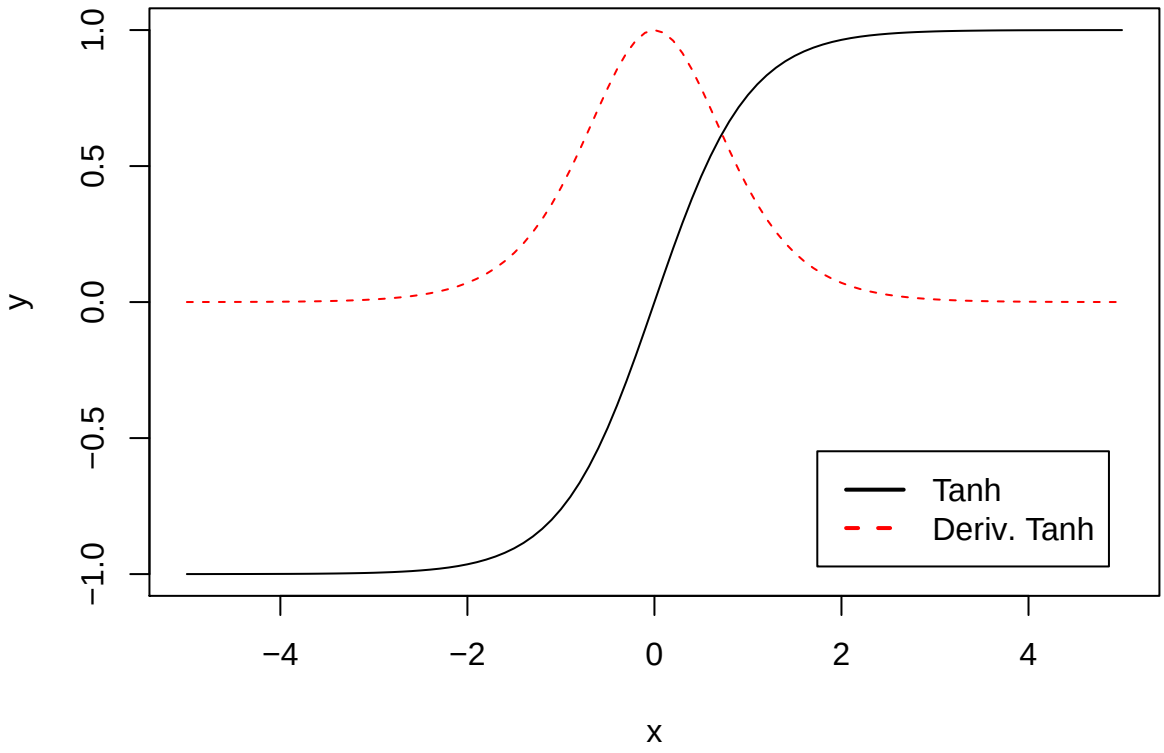


Figure 6.4: Htan Activation Function & Derivative

```
par(mar=c(5,4,2,2)+0.1)
act_tanh <- function(x) tanh(x)
deriv_tanh <- function(x) 1.0 - act_tanh(x)^2
plot(act_tanh,xlim=c(-5,5),ylim=c(-1,1),xlab="x",ylab="y")
plot(deriv_tanh,xlim=c(-5,5),ylim=c(-1,1),xlab="x",ylab="y",add=TRUE, lty=2, col="red")
legend("bottomright", inset=.05,
  c('Tanh','Deriv. Tanh'), lwd=2, lty=c(1, 2), col=c("black","red"))
```

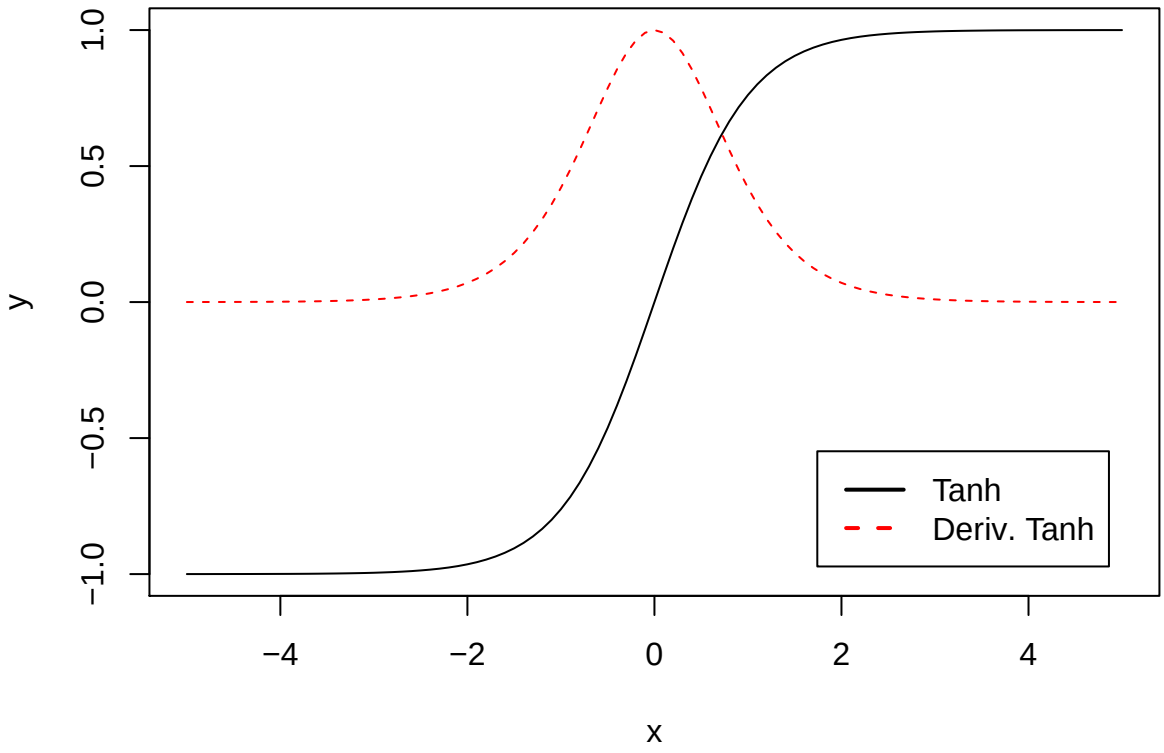
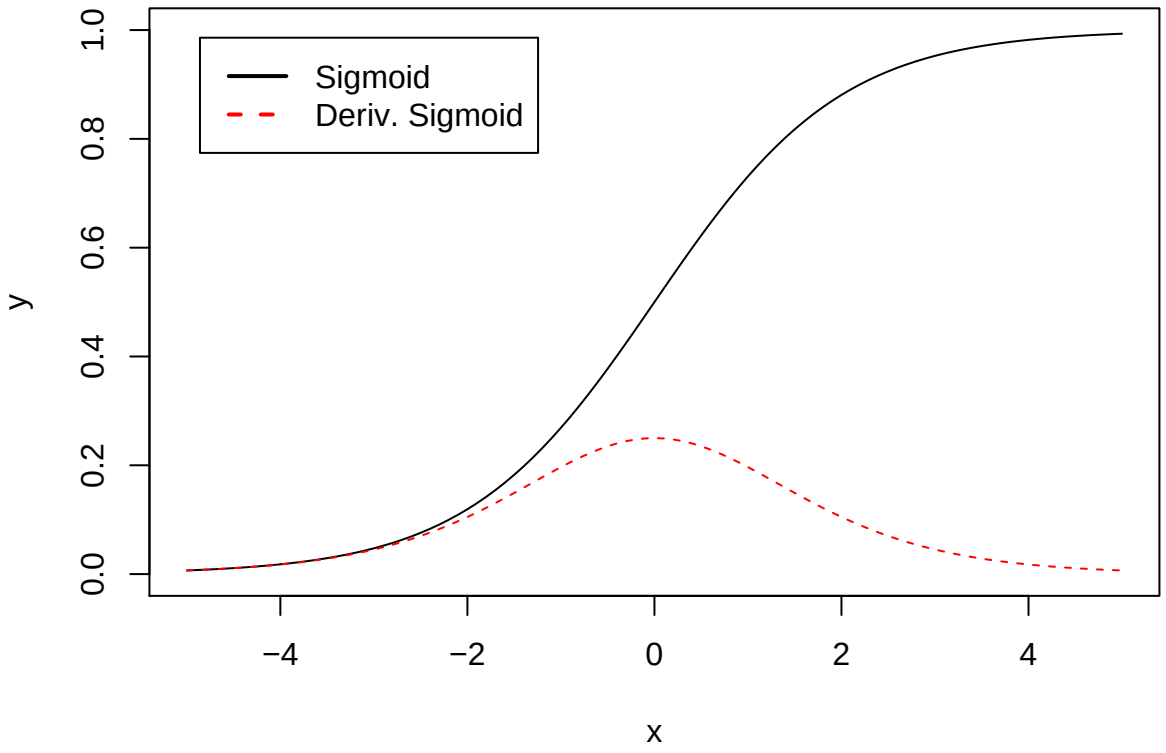


Figure 6.5: Sigmoid Activation Function & Derivative

```
par(mar=c(5,4,2,2)+0.1)
act_sigmoid <- function(x) 1/(1+exp(-x))
deriv_sigmoid <- function(x) act_sigmoid(x) * (1-act_sigmoid(x))
plot(act_sigmoid,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y")
plot(deriv_sigmoid,xlim=c(-5,5),ylim=c(0,1),xlab="x",ylab="y",lty=2,add=TRUE, col="red")
legend("topleft", inset=.05,
  c('Sigmoid','Deriv. Sigmoid'), lwd=2, lty=c(1, 2), col=c("black","red"))
```



Equation 6.13: Nesterov Momentum

$$n_0 = 0 \ , \ n_t = \alpha n_{t-1} + \epsilon \frac{\partial E}{\partial w_t}$$

Equation 6.14: Nesterov Update

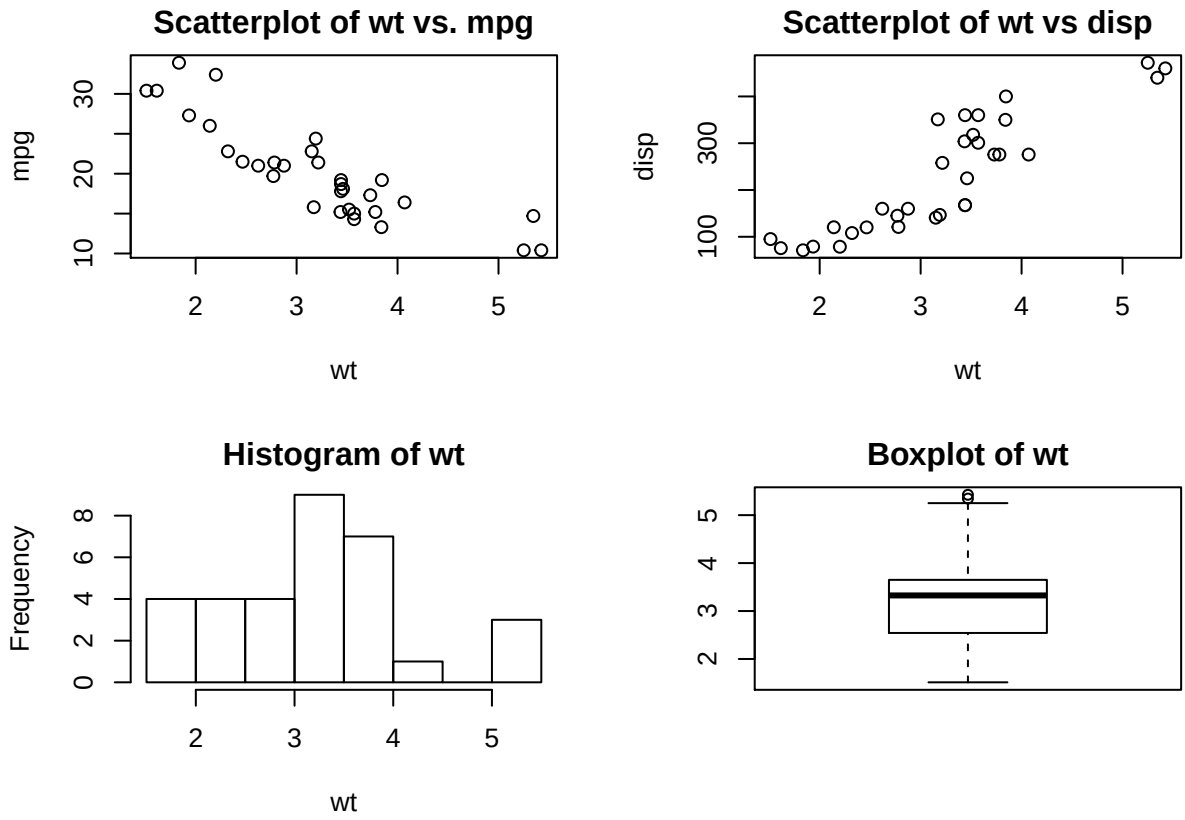
$$\Delta w_t = \alpha n_{t-1} - (1 + \alpha) n_t$$

Chapter 7: Other Propagation Training

Chapter 8: Chapter 8: NEAT, CPPN and HyperNEAT

Figure: HyperNEAT Activation Functions

```
par(mar=c(5,4,2,2)+0.1)
attach(mtcars)
par(mfrow=c(2,2))
plot(wt,mpg, main="Scatterplot of wt vs. mpg")
plot(wt,disp, main="Scatterplot of wt vs disp")
hist(wt, main="Histogram of wt")
boxplot(wt, main="Boxplot of wt")
```



Chapter 9: Deep Learining

Equation: Propagate Up

$$\bar{h}_i^+ = sigmoid(\sum_j w_j v_j + b_i)$$

Equation: Sample h+

$$h_i^+ = \begin{cases} 1 & r < \bar{h}_i^+ \\ 0 & r \geq \bar{h}_i^+ \end{cases}$$

Equation: Propagate Up

$$\bar{v}_i^- = sigmoid(\sum_j w_j h_j + b_i)$$

Equation: Sample h-

$$v_i^- = \begin{cases} 1 & r < \bar{v}_i^- \\ 0 & r \geq \bar{v}_i^- \end{cases}$$

Equation: Update weights

$$\Delta_{ij} = \frac{\epsilon(\bar{h}_i^+ x_j - \bar{h}_i^- v_j^-)}{|x|}$$

Equation: Update biases

$$\Delta b_i = \frac{\epsilon(h_i^+ - \bar{h}_i^-)}{|x|}$$

Chapter 11: Pruning and Model Selection

Chapter 12: Dropout and Regularization

Equation 7.1: L1 Error Term Objective

$$E_1 = \lambda_1 \sum_w |w|$$

Equation 7.2: L1 Error Term

$$E_1 = \frac{\lambda_1}{n} \sum_w |w|$$

Equation 7.3: L1 Weight Partial Derivative

$$\frac{\partial}{\partial w} E_1 = \frac{\lambda_1}{n} sgn(w)$$

Equation 7.4: L2 Error Term Objective

$$E_2 = \lambda_2 \sum_w w^2$$

Equation 7.5: L2 Error Term

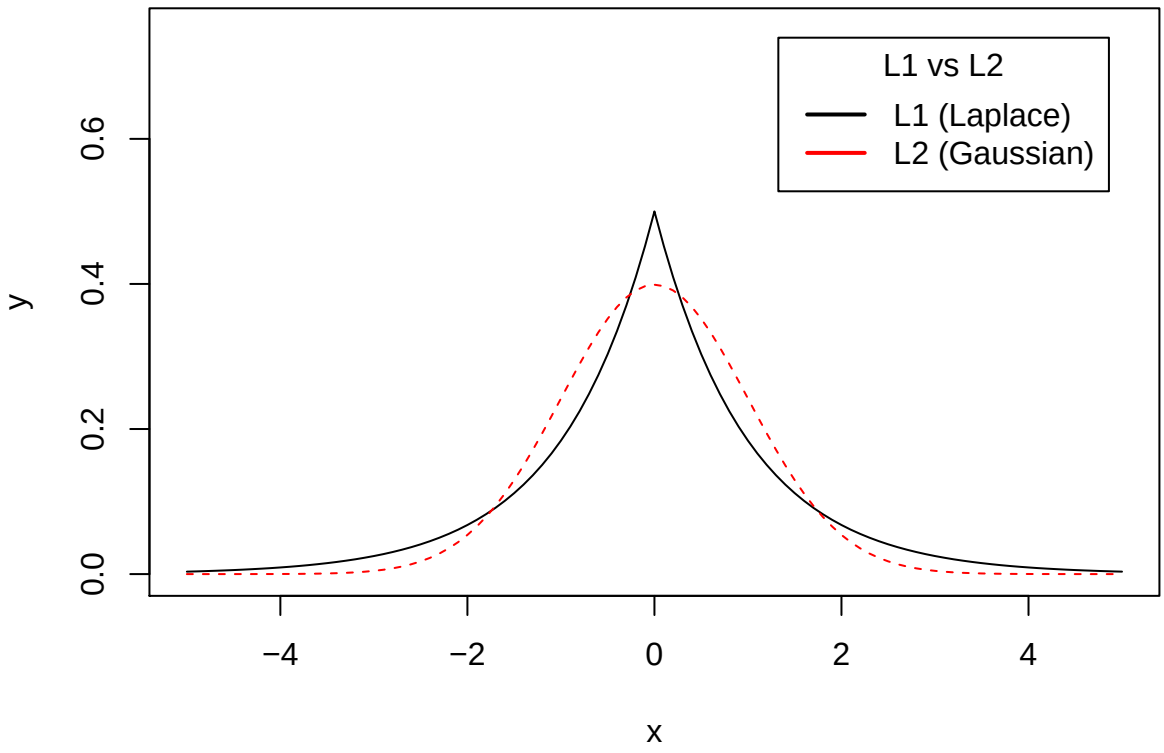
$$E_2 = \frac{\lambda_2}{n} \sum_w w^2$$

Equation 7.6: L2 Weight Partial Derivative

$$\frac{\partial}{\partial w} E_2 = \frac{\lambda_2}{n} w$$

Figure: Sigmoid Activation Function

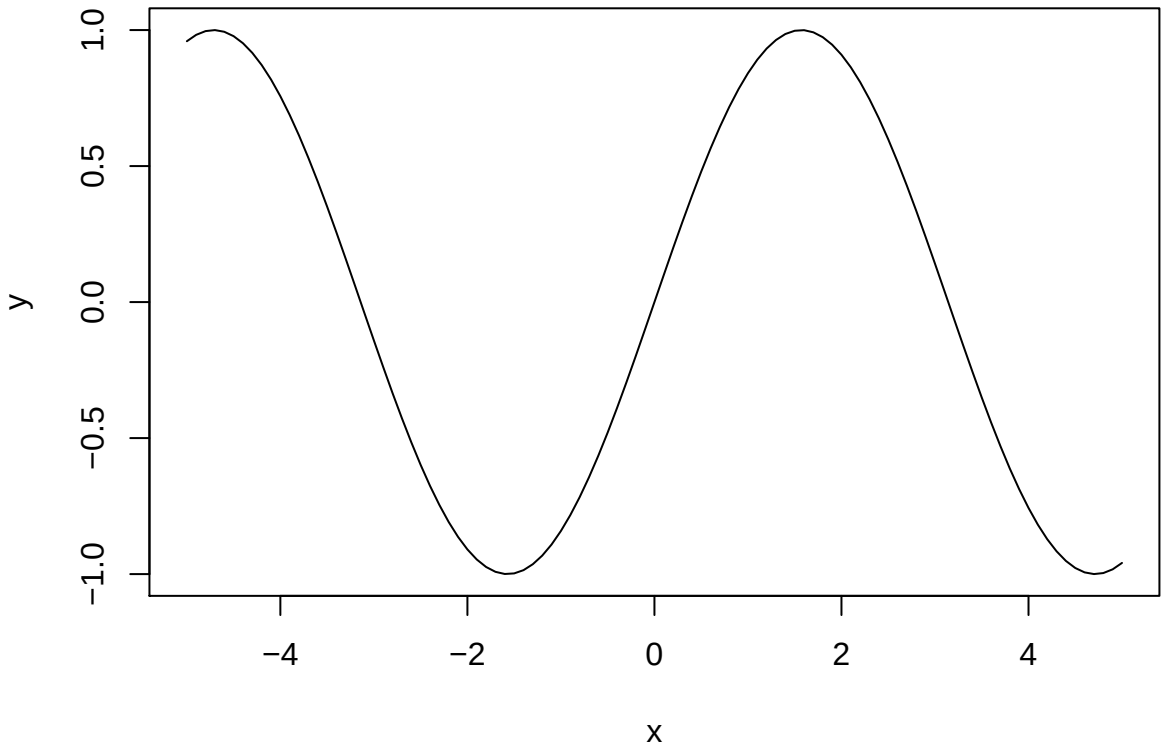
```r
par(mar=c(5,4,2,2)+0.1)
dist_laplace <- function(x) 1/(2*1)*exp(-abs(x-0)/1)
dist_gaussian <- function(x) dnorm(x)
plot(dist_laplace,xlim=c(-5,5),ylim=c(0,0.75),xlab="x",ylab="y")
plot(dist_gaussian,xlim=c(-5,5),ylim=c(0,0.75),xlab="x",ylab="y",add=TRUE, lty=2, col="red")
legend("topright", inset=.05, title="L1 vs L2",
  c('L1 (Laplace)','L2 (Gaussian)'), lwd=2, lty=c(1, 1, 1, 1, 2), col=c("black","red"))
```



Chapter 13: Time Series and Recurrent Networks

Figure: Sine Wave

```r
par(mar=c(5,4,2,2)+0.1)
plot(sin,xlim=c(-5,5),ylim=c(-1,1),xlab="x",ylab="y")
```



Chapter 14: Architecting Neural Networks

$$\epsilon = K(1-\alpha)$$

```
f <- function(k,alpha) cat(sprintf("k=%f, alpha=%f -> eta=%f\n", k, alpha,k*(1-alpha)))

f(0.5,0.2)
```

```
## k=0.500000, alpha=0.200000 -> eta=0.400000
```

```
f(0.5,0.3)
```

```
## k=0.500000, alpha=0.300000 -> eta=0.350000
```

```
f(0.5,0.4)
```

```
## k=0.500000, alpha=0.400000 -> eta=0.300000
```

```
f(1.0,0.2)
```

```
## k=1.000000, alpha=0.200000 -> eta=0.800000
```

```
f(1.0,0.3)
```

```
## k=1.000000, alpha=0.300000 -> eta=0.700000
```

```r
f(1.0,0.4)
```

```
## k=1.000000, alpha=0.400000 -> eta=0.600000
```

```r
f(1.5,0.2)
```

```
## k=1.500000, alpha=0.200000 -> eta=1.200000
```

```r
f(1.5,0.3)
```

```
## k=1.500000, alpha=0.300000 -> eta=1.050000
```

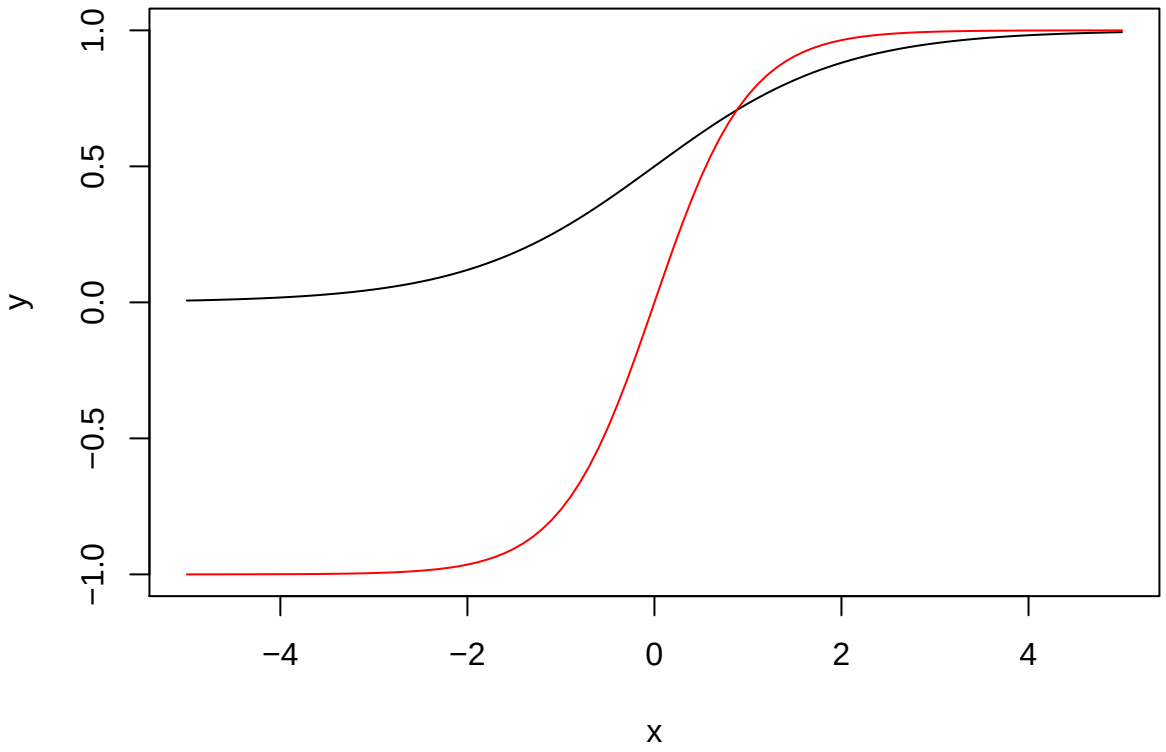
```r
f(1.5,0.4)
```

```
## k=1.500000, alpha=0.400000 -> eta=0.900000
```

$$\Delta w_{(t)} = \epsilon \frac{\partial E}{\partial w_{(t)}} + \alpha \Delta w_{(t-1)}$$

Figure: TanH Sigmoid Activation Function

```r
par(mar=c(5,4,2,2)+0.1)
sigmoid <- function(x) 1/(1+exp(-x))
tanh_shift <- function(x) tanh(x)
plot(sigmoid,xlim=c(-5,5),ylim=c(-1,1),xlab="x",ylab="y")
plot(tanh_shift,xlim=c(-5,5),ylim=c(-1,1),xlab="x",ylab="y",add=TRUE, col="red")
```



Chapter 15: Visualization

Chapter 16: Modeling with Neural Networks