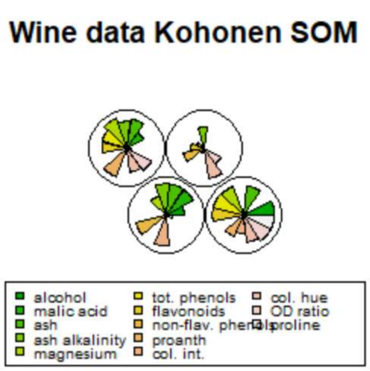
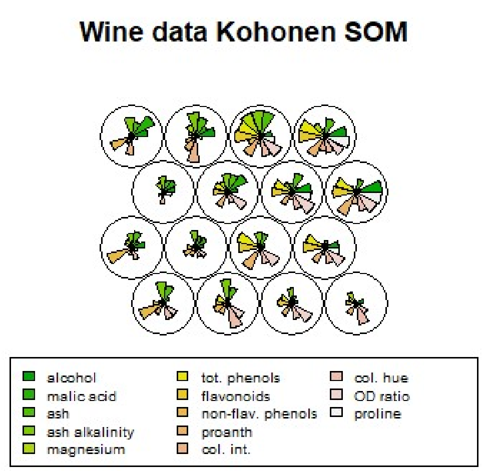
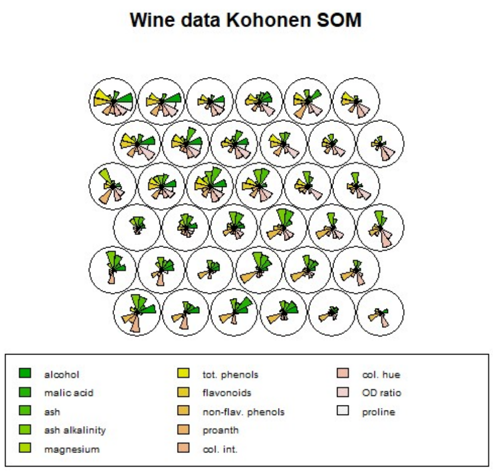
DL HW03 Kwak Yongha 곽용하 2014121047

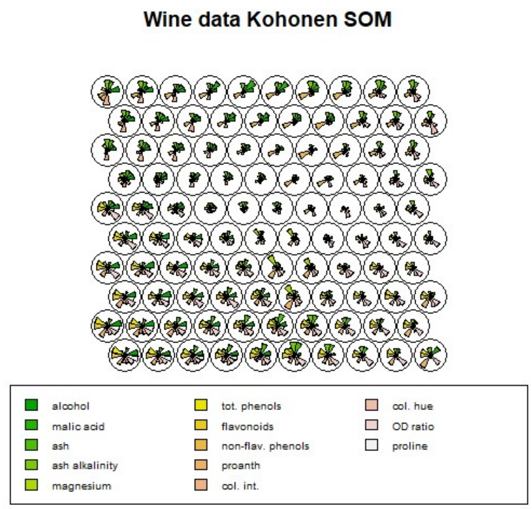
1.

```
library("kohonen")

data("wines")


set.seed(123)

som.wines1 = som(scale(wines), grid = somgrid(2, 2, "hexagonal"))

som.wines2 = som(scale(wines), grid = somgrid(4, 4, "hexagonal"))

som.wines3 = som(scale(wines), grid = somgrid(6, 6, "hexagonal"))

som.wines4 = som(scale(wines), grid = somgrid(10, 10, "hexagonal"))

som.wines14 = som(scale(wines), grid = somgrid(14, 14, "hexagonal"))


plot(som.wines1, main = "Wine data Kohonen SOM")

plot(som.wines2, main = "Wine data Kohonen SOM")

plot(som.wines3, main = "Wine data Kohonen SOM")

plot(som.wines4, main = "Wine data Kohonen SOM")

plot(som.wines14, main = "Wine data Kohonen SOM")
```









#Explain how it changes with increasing number of grids and what do you think is the appropriate number of grids?

결론) 6x6가 가장 적합하다고 생각합니다.

Kohonen SOM의 장점 중 하나는 neighboring neurons가 similar codebook을 가지고 있음을 시각적으로 보여주는 것인데, 2x2는 너무 작아서 구분이 너무 크게 되어서 분류의 기능을 못하고 있고, 4x4도 하나 정도의 neighboring neuron과만 유사성을 보여, SOM의 장점을 퇴색시키고 있습니다. 한편, 10x10의 경우 Kohonen map 자체가 input space를 small space로 mapping(차원 축소) 하기 위한 것인데 그 역할 자체를 제대로 하지 못합니다. 따라서 6x6가 가장 적합하다고 생각합니다.

Can you fit SOM for 14 × 14 grids for this example? If not why?

14x14로 할 때 다음의 에러가 발생합니다.

```
Error in sample.int(length(x), size, replace, prob) :
  cannot take a sample larger than the population when 'replace = FALSE'
```

마찬가지로 14x14의 경우, error 이유처럼 모집단보다 더 큰 sampling을 해야 해서 fitting이불가능합니다. 또한 마찬가지로 Kohonen map 자체가 input space를 small space로 mapping(차원 축소) 하기 위한 것인데 그 역할 자체를 제대로 하지 못합니다.

2.

(a)

```
##(a)##

### Initial settings

# Initialize parameters

w1 = 0.30

w2 = 0.10

w3 = 0.60

w4 = 0.40

w5 = 0.70

w6 = 0.50

w = c(w1, w2, w3, w4, w5, w6)   #vectors of weight parameters


#set bias terms' values, arbitrarily

b1 = 0.35

b2 = 0.60

b = c(b1, b2) #vector of bias parameters


# input and target values

input1 = 1.5

input2 = 0.5

input = c(input1, input2)


target1 = 1

target2 = 1

target = c(target1, target2)
```

```
### define functions

sigmoid = function(z){

  return( 1/(1+exp(-z)) )

}


#given 'input' set, Set forwardProp function

forwardProp = function(input, w, b){

  # input to hidden layer

  neth1 = w[1]*input[1] + b[1]

  neth2 = w[2]*input[2] + b[1]

  neth3 = w[3]*input[2] + b[1]

  outh1 = sigmoid(neth1)

  outh2 = sigmoid(neth2)

  outh3 = sigmoid(neth3)


  # hidden layer to output layer

  neto1 = w[4]*outh1 + w[6]*outh3 + b[2]

  neto2 = w[5]*outh2 + b[2]

  outo1 = sigmoid(neto1)

  outo2 = sigmoid(neto2)


  res = c(outh1, outh2, outh3, outo1, outo2)

  return(res)

}

# 참고

forwardProp(input,w,b)

[1] 0.6899745 0.5986877 0.6570105 0.7693235 0.7347936
```

(b)

```
### backward propagation

res = forwardProp(input, w, b)

outh1 = res[1]; outh2 = res[2]; outh3 = res[3]; outo1 = res[4]; outo2 = res[5]


## update w_4, w_5, w_6, b2

# compute dE_dw4

dE_dw4 = -( target[1] - outo1 )*outo1*(1-outo1)*outh1

# compute dE_dw5

dE_dw5 = -( target[2] - outo2 )*outo2*(1-outo2)*outh2

# compute dE_dw6

dE_dw6 = -( target[1] - outo1 )*outo1*(1-outo1)*outh3

# compute dE_db2

dE_db2 = -( target[1] - outo1 )*outo1*(1-outo1)*1 + -( target[2] - outo2 )*outo2*(1-outo2)*1


## update w_1, w_2, w_3, b1

# compute dE_douth1 first

dneto1_douth1 = w4

dE_douth1 = -( target[1] - outo1 )*outo1*(1-outo1)*dneto1_douth1

# compute dE_douth2 first

dneto2_douth2 = w5

dE_douth2 = -( target[2] - outo2 )*outo2*(1-outo2)*dneto1_douth1

# compute dE_douth3 first

dneto3_douth3 = w6

dE_douth3 = -( target[1] - outo1 )*outo1*(1-outo1)*dneto1_douth1

# compute dE_dw1

douth1_dneth1 = outh1*(1-outh1)
```

```
dneth1_dw1 = input[1]

dE_dw1 = dE_douth1*douth1_dneth1*dneth1_dw1


# compute dE_dw2

douth2_dneth2 = outh2*(1-outh2)

dneth2_dw2 = input[2]

dE_dw2 = dE_douth2*douth2_dneth2*dneth2_dw2

# compute dE_dw3

douth3_dneth3 = outh3*(1-outh3)

dneth3_dw3 = input[2]

dE_dw3 = dE_douth3*douth3_dneth3*dneth3_dw3

# compute dE_db1

dE_db1 = -( target[1] - outo1 )*outo1*(1-outo1)*dneto1_douth1*douth1_dneth1*1 + -( target[2] - outo2 )*outo2*(1-outo2)*dneto2_douth2*douth2_dneth2*1 +

   -( target[1] - outo1 )*outo1*(1-outo1)*dneto3_douth3*douth3_dneth3*1


### update all parameters via a gradient descent

w1 = w1 - gamma*dE_dw1

w2 = w2 - gamma*dE_dw2

w3 = w3 - gamma*dE_dw3

w4 = w4 - gamma*dE_dw4

w5 = w5 - gamma*dE_dw5

w6 = w6 - gamma*dE_dw6

b1 = b1 - gamma*dE_db1

b2 = b2 - gamma*dE_db2


w = c(w1, w2, w3, w4, w5, w6)

b = c(b1, b2)
```

```
#
w
[1] 0.3 0.1 0.6 0.4 0.7 0.5
b
[1] 0.35 0.60
```

(c)

```
##공통 코드## #gamma 값만 다르게 설정
### Implement Forward-backward propagation #1000번 반복 이런식으로 ㅇㅇ
err = c()
ts_w1 = c()
ts_w2 = c()
ts_w3 = c()
ts_w4 = c()
ts_w5 = c()
ts_w6 = c()
ts_b1 = c()
ts_b2 = c()

for(i in 1:1000){

  ###error function
  error = function(res, target){
    err = 0.5*(target[1] - res[4])^2 + 0.5*(target[2] - res[5])^2
    return(err)
  }
```

```
### forward
res = forwardProp(input, w, b)
outh1 = res[1]; outh2 = res[2]; outh3 = res[3]; outo1 = res[4]; outo2 = res[5]
### compute error
err[i] = error(res, target)
### trace all NN parameters
ts_w1[i] = w1
ts_w2[i] = w2
ts_w3[i] = w3
ts_w4[i] = w4
ts_w5[i] = w5
ts_w6[i] = w6
ts_b1[i] = b1
ts_b2[i] = b2

### backward propagation
## update w_4, w_5, w_6, b2
# compute dE_dw4
dE_dw4 = -( target[1] - outo1 )*outo1*(1-outo1)*outh1
# compute dE_dw5
dE_dw5 = -( target[2] - outo2 )*outo2*(1-outo2)*outh2
# compute dE_dw6
dE_dw6 = -( target[1] - outo1 )*outo1*(1-outo1)*outh3

# compute dE_db2
dE_db2 = -( target[1] - outo1 )*outo1*(1-outo1)*1 + -( target[2] - outo2 )*outo2*(1-outo2)*1
```

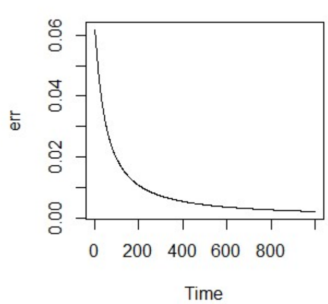
```
## update w_1, w_2, w_3, b1
# compute dE_douth1 first
dneto1_douth1 = w4
dE_douth1 = -( target[1] - outo1 )*outo1*(1-outo1)*dneto1_douth1
# compute dE_douth2 first
dneto2_douth2 = w5
dE_douth2 = -( target[2] - outo2 )*outo2*(1-outo2)*dneto1_douth1
# compute dE_douth3 first
dneto3_douth3 = w6
dE_douth3 = -( target[1] - outo1 )*outo1*(1-outo1)*dneto1_douth1


# compute dE_dw1
douth1_dneth1 = outh1*(1-outh1)
dneth1_dw1 = input[1]
dE_dw1 = dE_douth1*douth1_dneth1*dneth1_dw1
# compute dE_dw2
douth2_dneth2 = outh2*(1-outh2)
dneth2_dw2 = input[2]
dE_dw2 = dE_douth2*douth2_dneth2*dneth2_dw2
# compute dE_dw3
douth3_dneth3 = outh3*(1-outh3)
dneth3_dw3 = input[2]
dE_dw3 = dE_douth3*douth3_dneth3*dneth3_dw3


# compute dE_db1
dE_db1 = -( target[1] - outo1 )*outo1*(1-outo1)*dneto1_douth1*douth1_dneth1*1 + -( target[2] - outo2 )*outo2*(1-outo2)*dneto2_douth2*douth2_dneth2*1 +
```

```
    -( target[1] - outo1 )*outo1*(1-outo1)*dneto3_douth3*douth3_dneth3*1

  ### update all parameters via a gradient descent
  w1 = w1 - gamma*dE_dw1
  w2 = w2 - gamma*dE_dw2
  w3 = w3 - gamma*dE_dw3
  w4 = w4 - gamma*dE_dw4
  w5 = w5 - gamma*dE_dw5
  w6 = w6 - gamma*dE_dw6
  b1 = b1 - gamma*dE_db1
  b2 = b2 - gamma*dE_db2

  w = c(w1, w2, w3, w4, w5, w6)
  b = c(b1, b2)

}
```

##Draw error rate figure, and calculate prediction results as in the page 37-38 of the DL8 slide.

Case 1) gamma = 0.1

ts.plot(err)

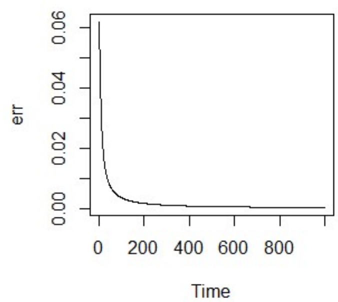


pred = forwardProp(input, w, b)

pred[4:5]

[1] 0.9636527 0.9472225

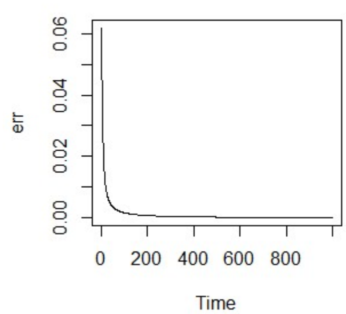
Case 2) gamma = 0.6



pred = forwardProp(input, w, b) ; pred[4:5]

[1] 0.9863896 0.9793567

Case3) gamma = 1.2



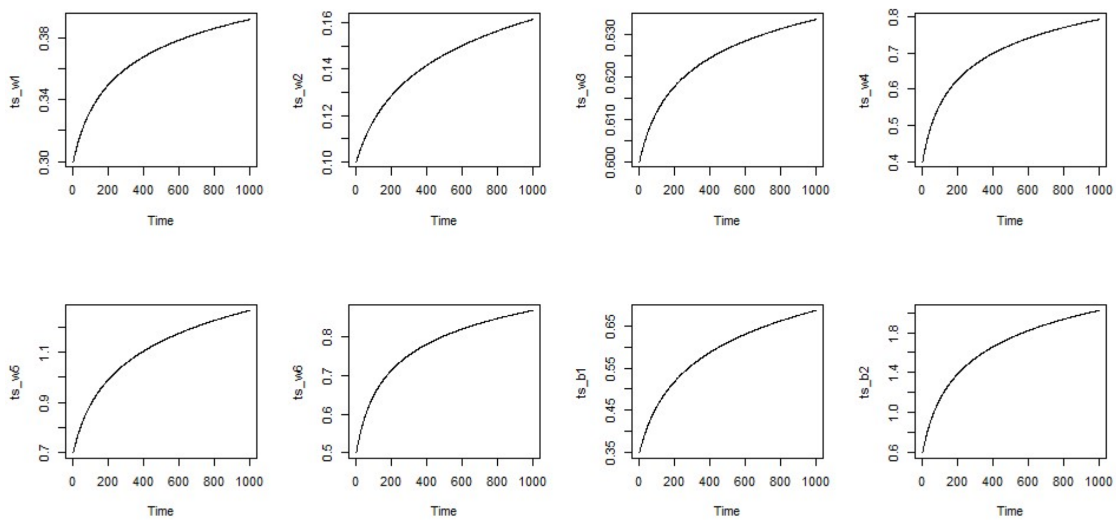
pred[4:5]

[1] 0.9905890 0.9856219

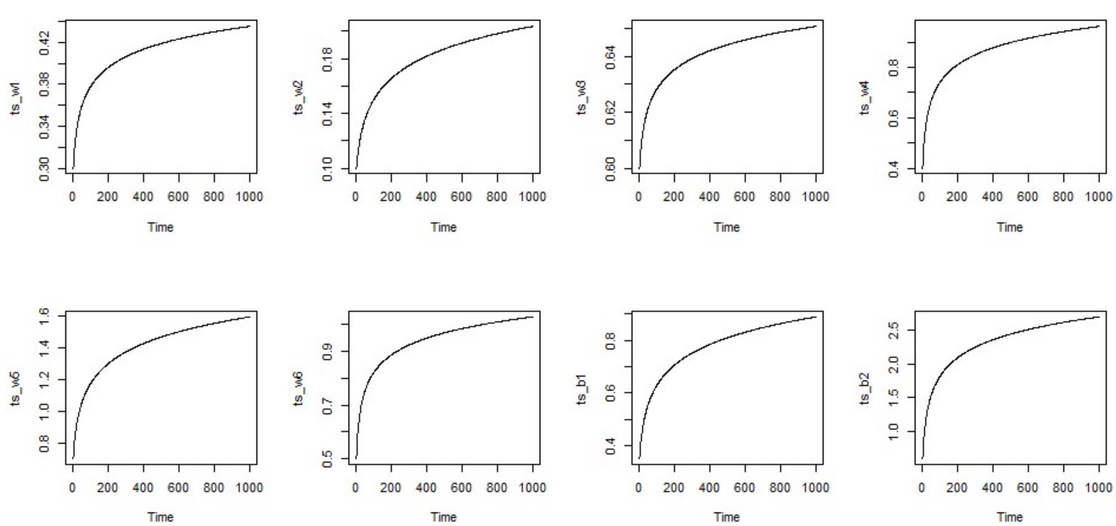
##Draw traceplots for all neural network parameters and report how they change with increasing iteration.

Case1) Gamma = 0.1

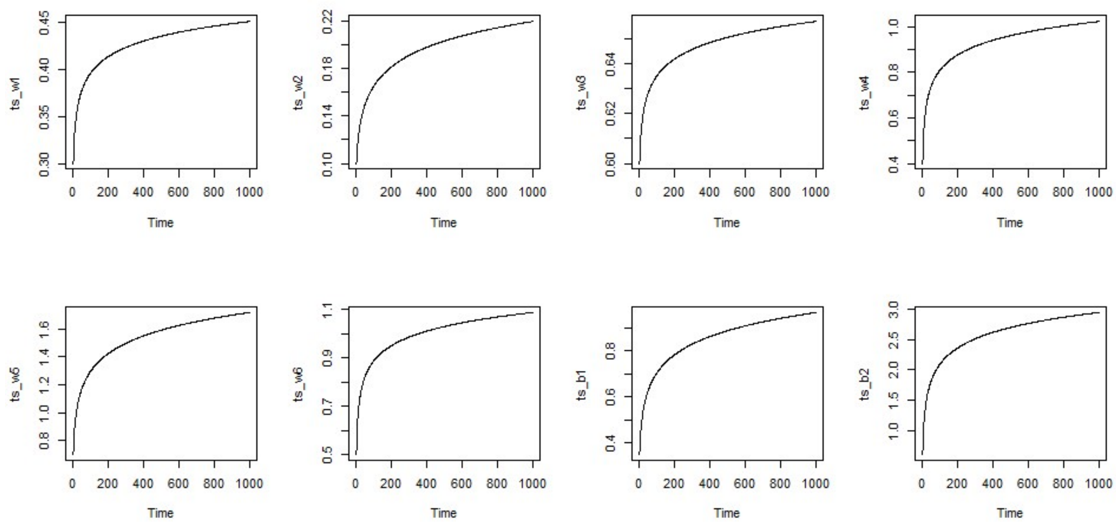
```
par(mfrow=c(2,4))

ts.plot(ts_w1) ; ts.plot(ts_w2) ; ts.plot(ts_w3) ; ts.plot(ts_w4)

ts.plot(ts_w5) ; ts.plot(ts_w6) ; ts.plot(ts_b1) ; ts.plot(ts_b2)
```



Case2) Gamma = 0.6



Case3) Gamma = 1.2



반복회수가 커짐에 따라 converging 하는 가속도가 초반에 빠르고 후반에는 느려진다.

#Discuss about the convergence of neural network fittings based on different learning rates.

Gamma 값이 커짐에 따라 converging 하는 가속도가 초반에 더 빠르고 후반에는 느려진다. 즉, 반복회수가 증가함에 따라 나타나는 현상의 특징이 강화되는 모습을 보인다. 이는 error 값의 trace가 마찬가지로 gamma 값이 클수록 초반에 급격히 감소한 후 뒤로 갈수록 완만해지는 모습과 유관하다.

한편 수렴 값의 경우 gamma 값이 커짐에 따라 전반적으로 상승하는 경향이 있었다.