Supervised Artificial Neural Network with Selective Connection based on Gene Patterns

Zihang Zeng

ANN-SCGP proposes a supervised Artificial Neural Network with Selective Connection based on Gene Patterns (ANN-SCGP) model using Omics-data

Data preparation

Loading example

```
##Loading example
#data=read.csv("data.csv")
#clin=read.csv("clin.csv")
#validate_data=read.csv("validate_data.csv")
#validate_clin=read.csv("validate_clin.csv")
library(ANNSCGP)
library(ggplot2)
metabric <- hdf5r::H5File$new("data/metabric_IHC4_clinical_train_test.h5", mode = "r")
metabric
```

```
## Class: H5File
## Filename: C:\Users\zZ\Documents\ANNSCGP\data\metabric_IHC4_clinical_train_test.h5
## Access type: H5F_ACC_RDONLY
## Listing:
##   name  obj_type dataset.dims dataset.type_class
##   test H5I_GROUP         <NA>               <NA>
##  train H5I_GROUP         <NA>               <NA>
```

```
###training data
data=t(metabric[["train"]][["x"]][,])
rownames(data)=1:nrow(data)
colnames(data)=1:ncol(data)
clin=data.frame(time=metabric[["train"]][["t"]][]/100,status=metabric[["train"]][["e"]][])
rownames(clin)=1:nrow(clin)
data[1:5,1:5]
```

```
##          1        2         3        4 5
## 1 5.603834 7.811392 10.797988 5.967607 1
## 2 5.284882 9.581043 10.204620 5.664970 1
## 3 5.920251 6.776564 12.431715 5.873857 0
## 4 6.654017 5.341846  8.646379 5.655888 0
## 5 5.456747 5.339741 10.555724 6.008429 1
```

```
clin[1:5,]
```

```
##        time status
## 1 0.9933334      0
## 2 0.9573333      1
## 3 1.4023334      0
## 4 2.3930000      0
## 5 0.5693333      1
```

```
###testing data
validate_data=t(metabric[["test"]][["x"]][,])
colnames(validate_data)=1:ncol(validate_data)
rownames(validate_data)=1:nrow(validate_data)
validate_clin=data.frame(time=metabric[["test"]][["t"]][]/100,status=metabric[["test"]][["e"]][])
rownames(validate_clin)=1:nrow(validate_clin)
validate_data[1:5,1:5]
```

```
##          1        2         3        4 5
## 1 8.003323 5.383952 13.391568 6.177617 0
## 2 5.877926 8.036838 10.164740 5.866741 0
## 3 5.892256 6.425127 11.480591 5.551163 1
## 4 5.556709 7.545435 11.240383 6.057502 0
## 5 5.501077 7.791979  9.923388 5.588843 1
```

```
validate_clin[1:5,]
```

```
##        time status
## 1 1.0210000      0
## 2 1.7110001      0
## 3 1.7483333      0
## 4 1.0726667      1
## 5 0.4683333      1
```

```
###standardization
normalize_minmax<-function(x){
     return((x-min(x)+0.001)/(max(x)-min(x)+0.001))}
data=normalize_minmax(scale(data))
validate_data=normalize_minmax(scale(validate_data))
data[1:5,1:5]
```

```
##           1         2         3         4         5
## 1 0.1965050 0.3709469 0.2537325 0.2746444 0.3145287
## 2 0.1675138 0.5060012 0.2197192 0.2053290 0.3145287
## 3 0.2252658 0.2919720 0.3473820 0.2531721 0.1552515
## 4 0.2919616 0.1824787 0.1303967 0.2032488 0.1552515
## 5 0.1831355 0.1823182 0.2398453 0.2839941 0.3145287
```

```
validate_data[1:5,1:5]
```

```
##           1         2         3         4         5
## 1 0.4581029 0.1939577 0.4498491 0.3452831 0.1368032
## 2 0.2330192 0.4379694 0.2256232 0.2606435 0.1368032
## 3 0.2345368 0.2897247 0.3170591 0.1747237 0.3355397
## 4 0.1990017 0.3927704 0.3003676 0.3125806 0.1368032
## 5 0.1931101 0.4154474 0.2088522 0.1849827 0.3355397
```

Construction of selectively connected matrix (SCM)

Building SCM from inputs

```
library(philentropy)
library(NMF,quietly=TRUE)
```

```
## NMF - BioConductor layer [OK] | Shared memory capabilities [NO: windows] | Cores 15/16
```

```
library(reshape2)
n=5 ##n isthe number of L1 nodes
SCM.r <- NMF::nmf(t(data)+0.0000000001,n, nrun=2, seed=123456,.options='v1p1')
```

```
## NMF algorithm: 'brunet'
```

```
## Multiple runs: 2
```

```
## Mode: sequential [foreach:doParallelSNOW]
```

```
##
Runs: |
Runs: |                                                  |   0%
Runs: |
Runs: |==================================================| 100%
## System time:
##  用户  系统  流逝
##  0.80  0.06 95.26
```

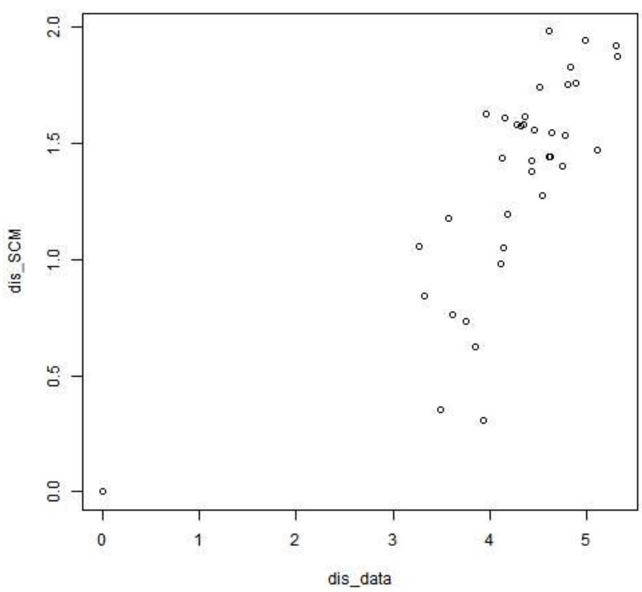
```
dis_data=melt(distance(t(data), method = "euclidean",test.na = F,use.row.names=T))[,3]
```

```
## Metric: 'euclidean'; comparing: 9 vectors.
```

```
dis_SCM=melt(distance(basis(SCM.r), method = "euclidean",test.na = F,use.row.names=T))[,3]
```

```
## Metric: 'euclidean'; comparing: 9 vectors.
```

```
plot(dis_data,dis_SCM)
```

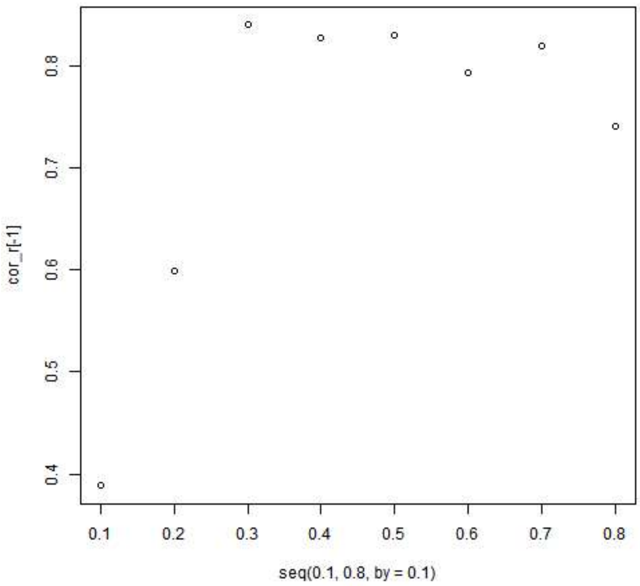


```
cor_r=NA
cor_p=NA
#Por is sparsity
for (Por in seq(0.1,0.8,by=0.1)){
weight_selective=basis(SCM.r)
weight_selective[(weight_selective)>quantile(melt(basis(SCM.r))[,3],Por) ]=1
weight_selective[(weight_selective)<=quantile(melt(basis(SCM.r))[,3],Por) ]=0
a2=melt(distance(weight_selective, method = "euclidean",test.na = F,use.row.names=T))[,3]
COR=cor.test(dis_data,a2)
cor_r=c(cor_r,COR$estimate)
cor_p=c(cor_p,COR$p.value)
}
```

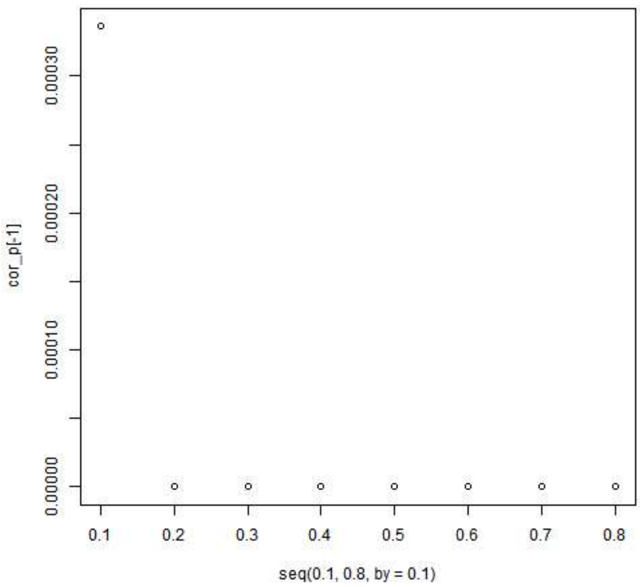
```
## Metric: 'euclidean'; comparing: 9 vectors.
```

```
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
```

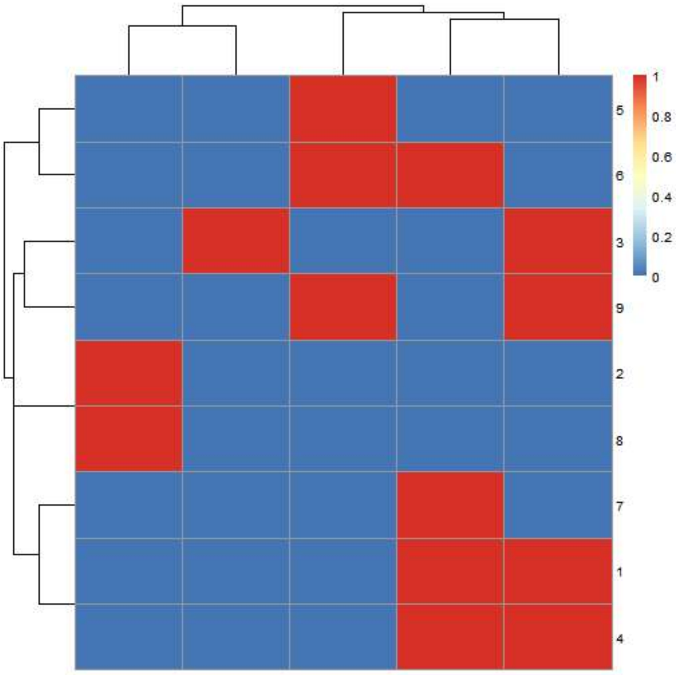
```
plot(seq(0.1,0.8,by=0.1),cor_r[-1])
```



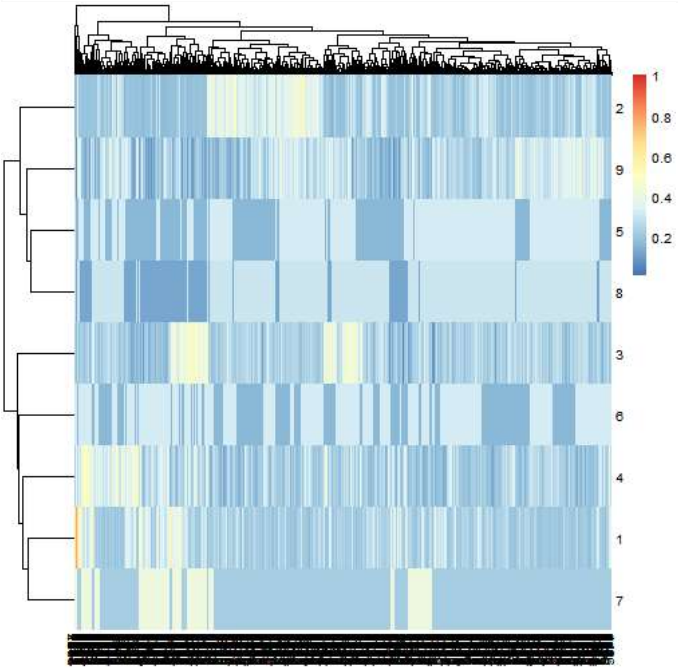

```
plot(seq(0.1,0.8,by=0.1),cor_p[-1])
```




```
Por=0.7 ##we choose the largest possible sparsity that does not cause the above cor_r to decrease rapidly
weight_selective=basis(SCM.r)
weight_selective[(weight_selective)>quantile(melt(basis(SCM.r))[,3],Por) ]=1
weight_selective[(weight_selective)<=quantile(melt(basis(SCM.r))[,3],Por) ]=0
pheatmap::pheatmap(weight_selective) ##weight_selective is SCM
```



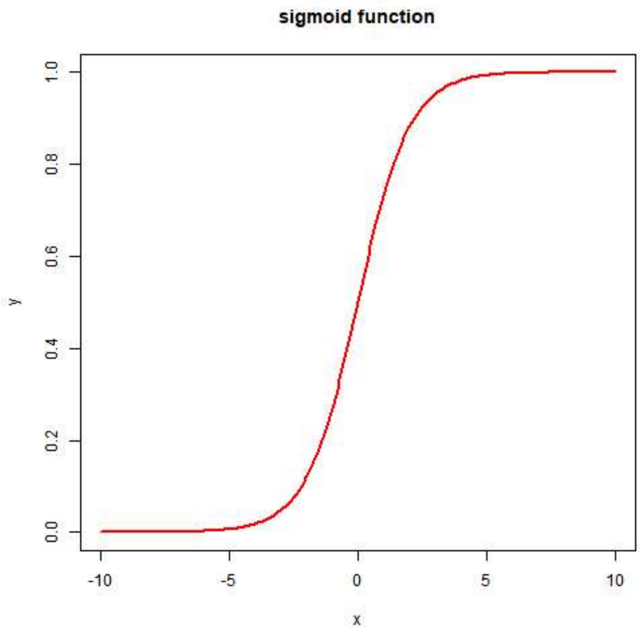

```
pheatmap::pheatmap(t(data))
```



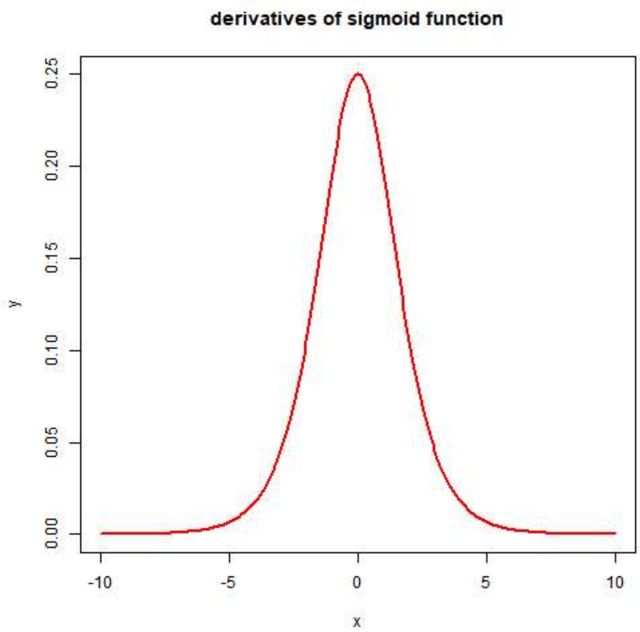
Activation function

sigmoid function

```
sigmoid<-function(x){
  return(1/(1+exp(-x)))
}
x=seq(-10,10,0.1)
y=sigmoid(x)
plot(x,y,type="l", lwd=2,col="red",main="sigmoid function")
```

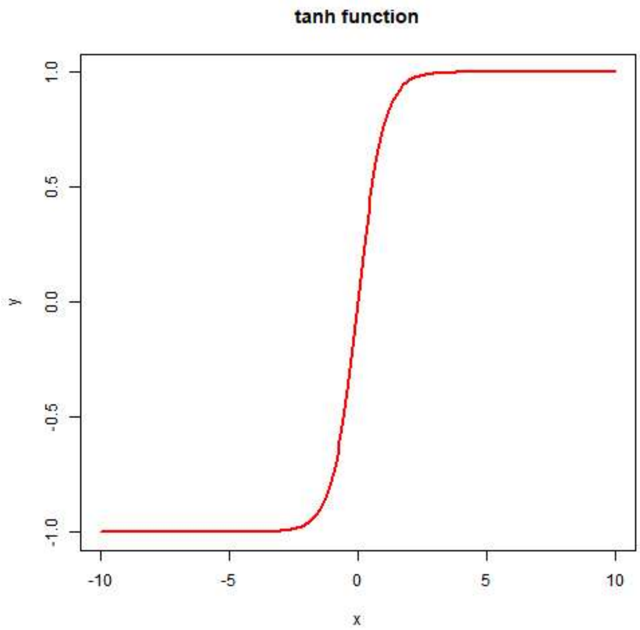



```
dev_sigmoid<-function(x){
  x=sigmoid(x)*(1-sigmoid(x))
  return(x)
}
y=dev_sigmoid(x)
plot(x,y,type="l", lwd=2,col="red",main="derivatives of sigmoid function")
```

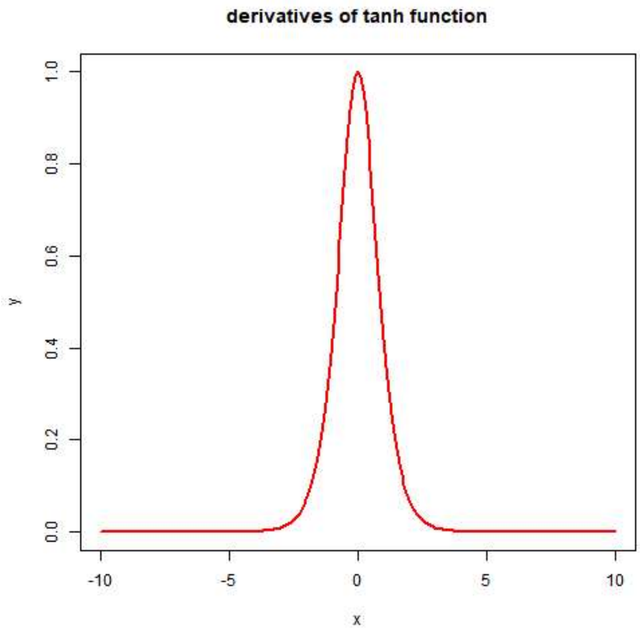


tanh function

```
tanh<-function(x){
  return((exp(x)-exp(-x))/(exp(x)+exp(-x)))
}
x=seq(-10,10,0.1)
y=tanh(x)
plot(x,y,type="l", lwd=2,col="red",main="tanh function")
```

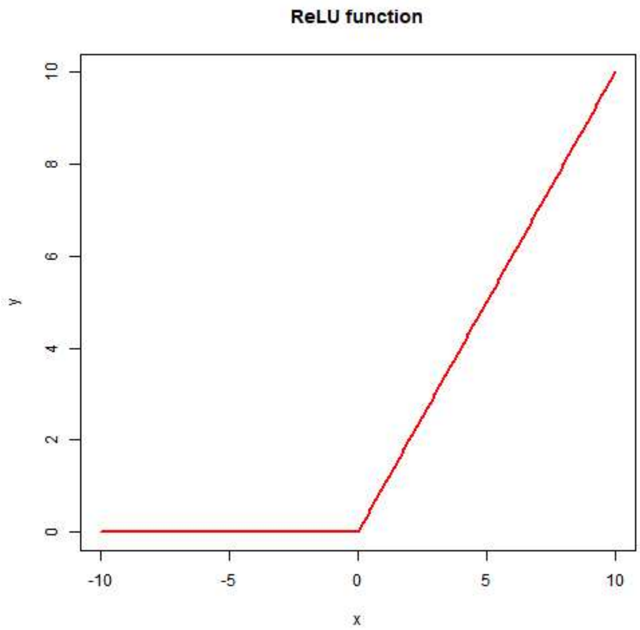



```
dev_tanh<-function(x){
  x=1-(tanh(x)^2)
  return(x)
}
y=dev_tanh(x)
plot(x,y,type="l", lwd=2,col="red",main="derivatives of tanh function")
```

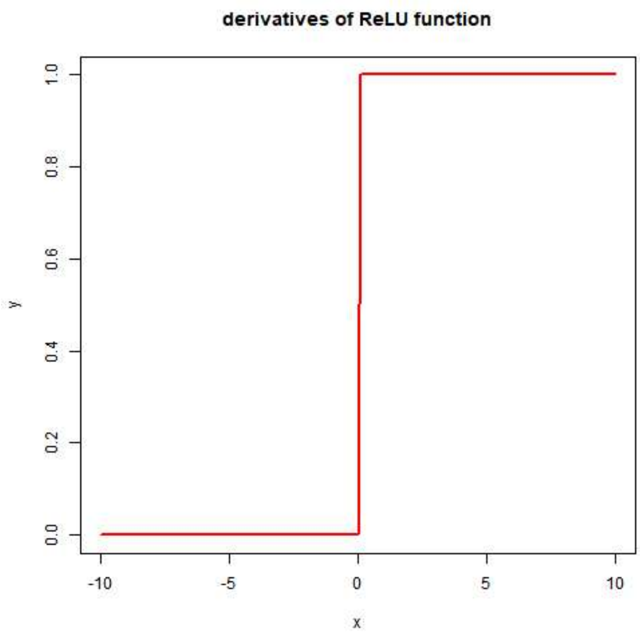


ReLU function

```
ReLU<-function(x){
  x[x<=0]=0
  return(x)
}
x=seq(-10,10,0.1)
y=ReLU(x)
plot(x,y,type="l", lwd=2,col="red",main="ReLU function")
```

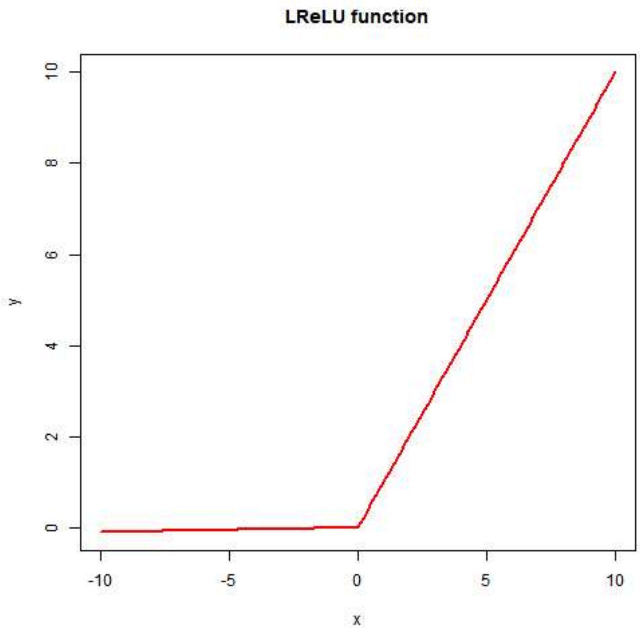



```
dev_ReLU<-function(x){
  x[x<=0]=0
  x[x>0]=1
  return(x)
}
y=dev_ReLU(x)
plot(x,y,type="l", lwd=2,col="red",main="derivatives of ReLU function")
```

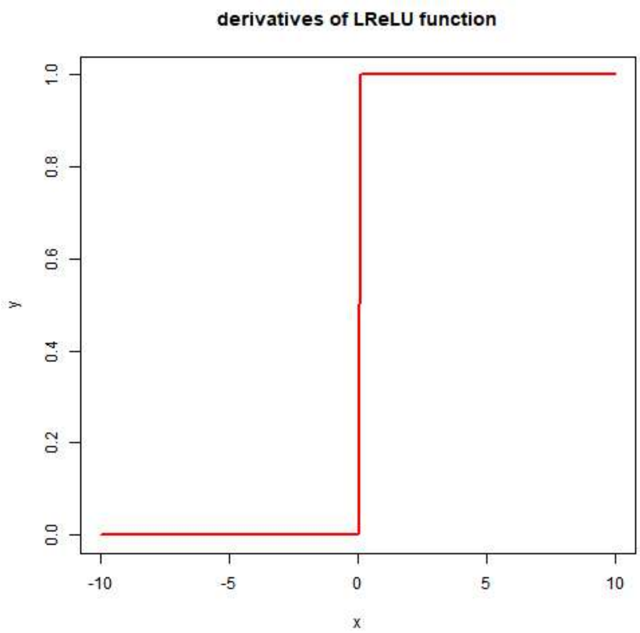


LReLU function

```
alpha=0.01
LReLU<-function(x){
  x[x<=0]=alpha*x[x<=0]
  return(x)
}
x=seq(-10,10,0.1)
y=LReLU(x)
plot(x,y,type="l", lwd=2,col="red",main="LReLU function")
```

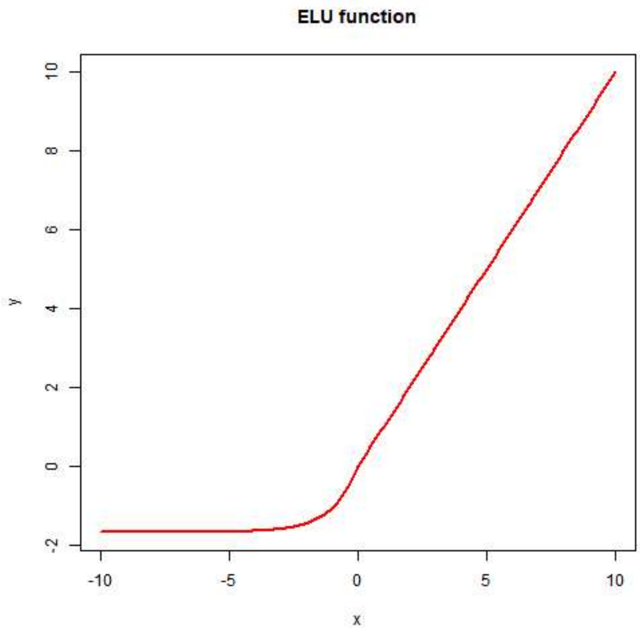



```
dev_LReLU<-function(x){
  x[x<=0]=alpha
  x[x>0]=1
  return(x)
}
y=dev_ReLU(x)
plot(x,y,type="l", lwd=2,col="red",main="derivatives of LReLU function")
```

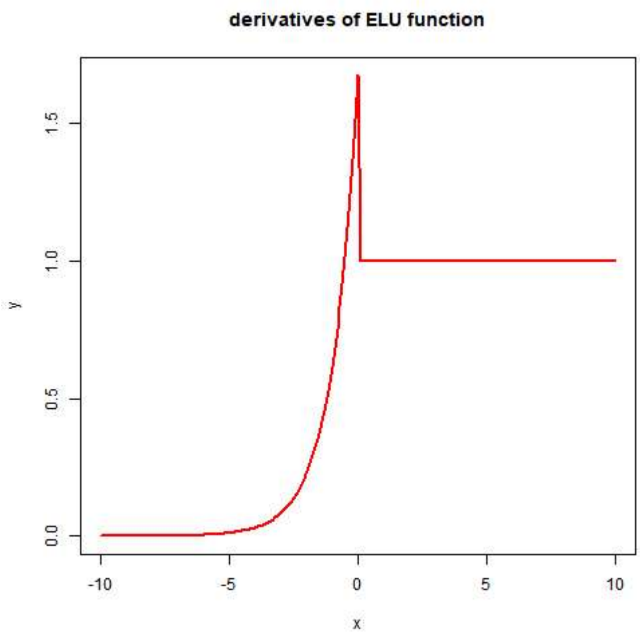


ELU function

```
alpha=1.67326324235
ELU<-function(x){
  x[x<=0]=alpha*(exp(x[x<=0])-1)
  return(x)
}
x=seq(-10,10,0.1)
y=ELU(x)
plot(x,y,type="l", lwd=2,col="red",main="ELU function")
```

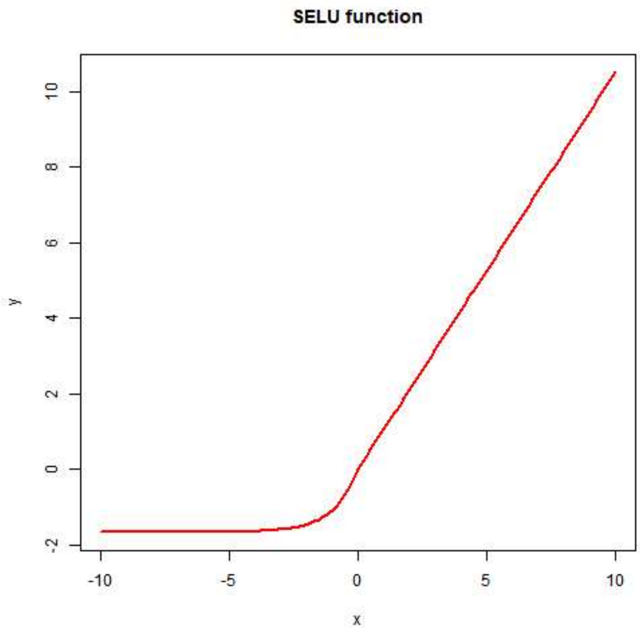



```
dev_ELU<-function(x){
  x1=x
  x1[x>0]=1
  x1[x<=0]=ELU(x)[x<=0]+alpha
  return(x1)
}
y=dev_ELU(x)
plot(x,y,type="l", lwd=2,col="red",main="derivatives of ELU function")
```

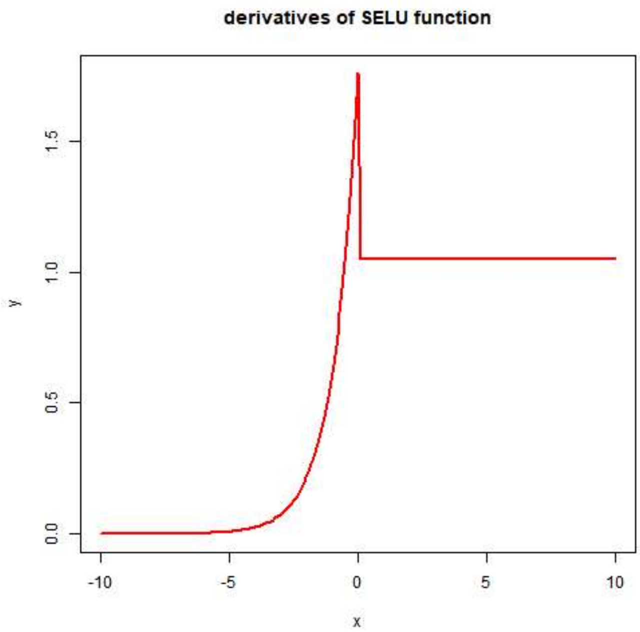


SELU function

```
alpha=1.67326324235
lambda=1.05070098736
SELU<-function(x){
  x=x*lambda
  x[x<=0]=alpha*(exp(x[x<=0])-1)
  return(x)
}
x=seq(-10,10,0.1)
y=SELU(x)
plot(x,y,type="l", lwd=2,col="red",main="SELU function")
```

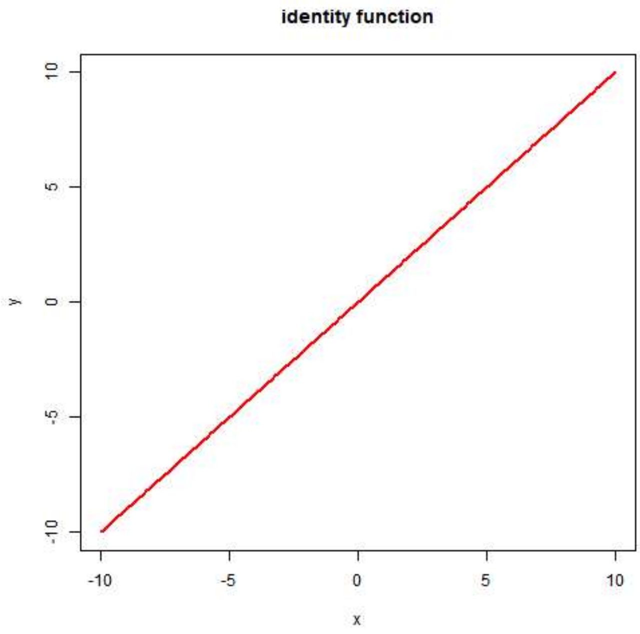



```
dev_SELU<-function(x){
  x1=x
  x1[x>0]=lambda
  x1[x<=0]=(SELU(x)[x<=0]+alpha)*lambda
  return(x1)
}
y=dev_SELU(x)
plot(x,y,type="l", lwd=2,col="red",main="derivatives of SELU function")
```

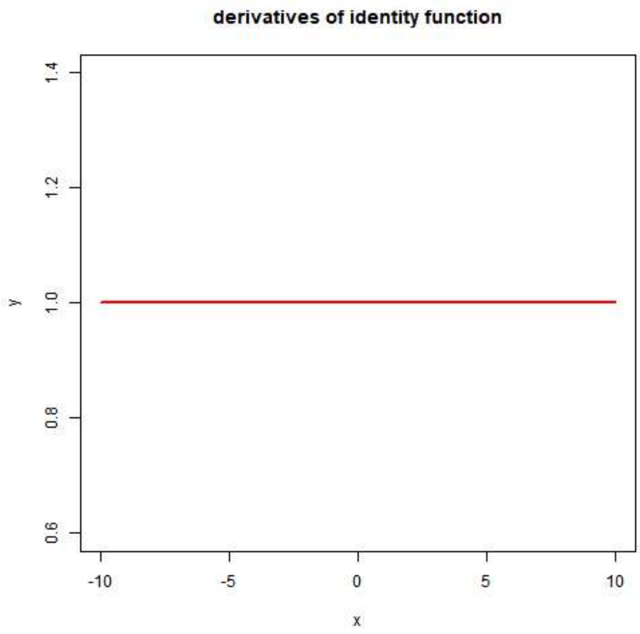


identity function

```
alpha=1
identity<-function(x){return(alpha*x)}
x=seq(-10,10,0.1)
y=identity(x)
plot(x,y,type="l", lwd=2,col="red",main="identity function")
```

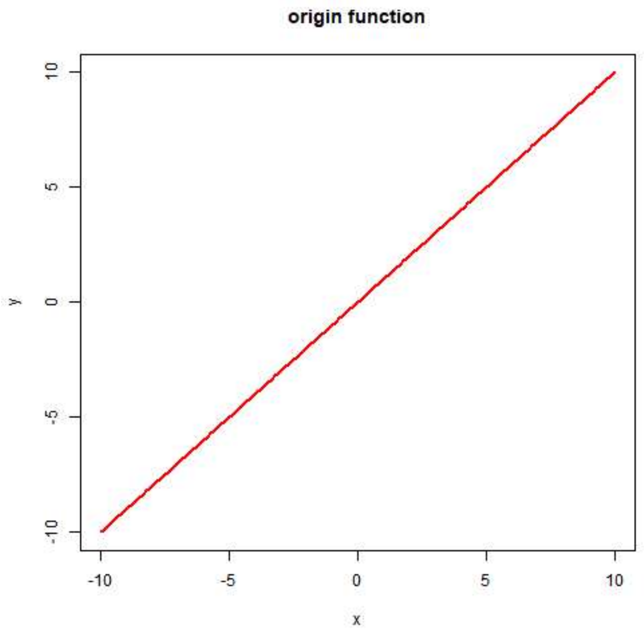



```
dev_identity<-function(x){return(rep(alpha,length(x)))}
y=dev_identity(x)
plot(x,y,type="l", lwd=2,col="red",main="derivatives of identity function")
```

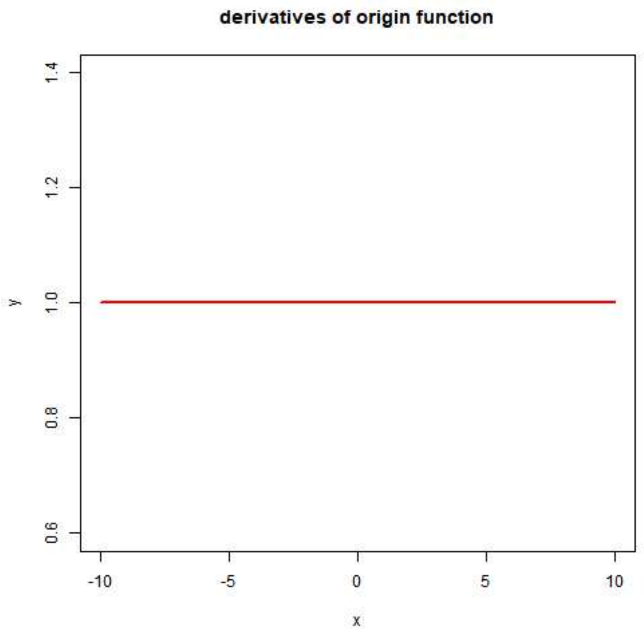


origin function

```
origin<-function(x){return(x)}
x=seq(-10,10,0.1)
y=origin(x)
plot(x,y,type="l", lwd=2,col="red",main="origin function")
```




```
dev_origin<-function(x){return(rep(1,length(x)))}
y=dev_origin(x)
plot(x,y,type="l", lwd=2,col="red",main="derivatives of origin function")
```



Parameter setting

```
parameters=list(alpha=1.67326324235,  #alpha value of activation function
                lambda=1.05070098736,   #lambda value of activation function
                learning_rate=0.0001, #learning rate
                acf=SELU, #activation function
                outf=exp, #activation function in output layer
                derivatives_outf=exp, #derivatives of activation function in output layer
                derivatives_acf=dev_SELU, #derivatives of activation function
                training_n=2000, #iterations
                message_n=100, #message after n iterations
                termination_threshold=0.95, #termination threshold
                greater_than_termination_threshold=F) #T: Terminate when error value is greater than termination threshold)
```

Training

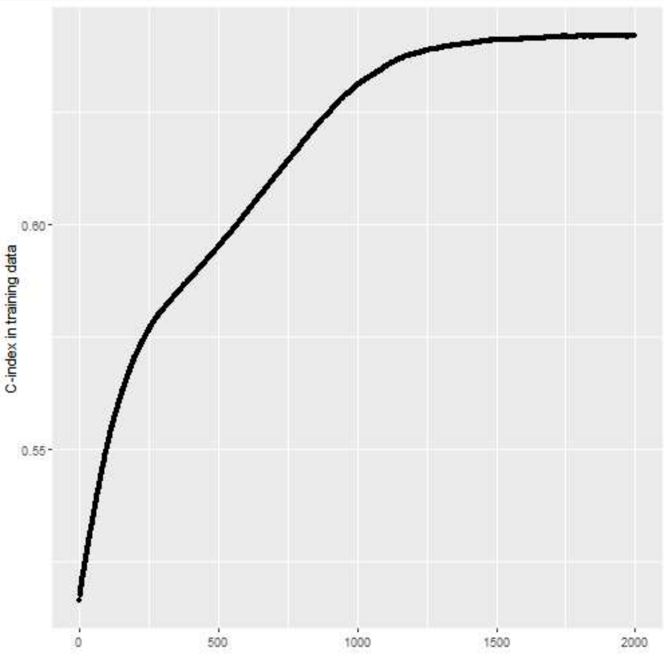
```
##option setting: A list containing the number of nodes per cluster in each layer (Start with the second layer)
option=list(c(5),c(1)) #there are five nodes in the 2nd and one nodes in the 3rd layer

SCGP_res=SCGP_train(data,clin,option,SCM=weight_selective,parameters=parameters,validate_data,validate_clin,survival_analyis=T)  ###validate_data anc
```

```
## 载入需要的程辑包：survival
```

```
## [1] "The iteration is 100"
## [1] "The error value is 0.550644439690511"
## [1] "The validate error value is 0.558024538100263"
## [1] "The iteration is 200"
## [1] "The error value is 0.570538415975526"
## [1] "The validate error value is 0.576666045972731"
## [1] "The iteration is 300"
## [1] "The error value is 0.58095159190144"
## [1] "The validate error value is 0.588066124593963"
## [1] "The iteration is 400"
## [1] "The error value is 0.58785537544192"
## [1] "The validate error value is 0.59747998262057"
## [1] "The iteration is 500"
## [1] "The error value is 0.594961956851044"
## [1] "The validate error value is 0.605942109945586"
## [1] "The iteration is 600"
## [1] "The error value is 0.602498865444851"
## [1] "The validate error value is 0.613224918792543"
## [1] "The iteration is 700"
## [1] "The error value is 0.610243518196781"
## [1] "The validate error value is 0.621542217532535"
## [1] "The iteration is 800"
## [1] "The error value is 0.617896664593347"
## [1] "The validate error value is 0.628721577390189"
## [1] "The iteration is 900"
## [1] "The error value is 0.624926578515544"
## [1] "The validate error value is 0.633935406451079"
## [1] "The iteration is 1000"
## [1] "The error value is 0.631046375173584"
## [1] "The validate error value is 0.636542320981524"
## [1] "The iteration is 1100"
## [1] "The error value is 0.635088730250394"
## [1] "The validate error value is 0.63703887613018"
## [1] "The iteration is 1200"
## [1] "The error value is 0.637835157483674"
## [1] "The validate error value is 0.636252663811474"
## [1] "The iteration is 1300"
## [1] "The error value is 0.639395711814336"
## [1] "The validate error value is 0.635383692301326"
## [1] "The iteration is 1400"
## [1] "The error value is 0.640305829078494"
## [1] "The validate error value is 0.634949206546252"
## [1] "The iteration is 1500"
## [1] "The error value is 0.641071267375388"
## [1] "The validate error value is 0.63366643907889"
## [1] "The iteration is 1600"
## [1] "The error value is 0.641307452698015"
## [1] "The validate error value is 0.632528500196553"
## [1] "The iteration is 1700"
## [1] "The error value is 0.641704392428715"
## [1] "The validate error value is 0.632652638983717"
## [1] "The iteration is 1800"
## [1] "The error value is 0.641910900014468"
## [1] "The validate error value is 0.632321602217946"
## [1] "The iteration is 1900"
## [1] "The error value is 0.641962836053999"
## [1] "The validate error value is 0.631452630707798"
## [1] "The iteration is 2000"
## [1] "The error value is 0.641903480580249"
## [1] "The validate error value is 0.631369871516355"
```

```
####C-index in training data
ggplot(NULL,aes(y=SCGP_res$cost,x=1:length(SCGP_res$cost)))+geom_point()+labs(x="",y="C-index in training data")
```



Get output value

```
library(survival)
output=SCGP_get_output(data,SCGP_res)
summary(coxph(Surv(clin$time,clin$status)~output))
```

```
## Call:
## coxph(formula = Surv(clin$time, clin$status) ~ output)
##
##   n= 1523, number of events= 887
##
##           coef exp(coef) se(coef)      z Pr(>|z|)
## output -1.6847    0.1855   0.1324 -12.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##        exp(coef) exp(-coef) lower .95 upper .95
## output    0.1855      5.391    0.1431    0.2405
##
## Concordance= 0.642  (se = 0.01 )
## Likelihood ratio test= 180.4  on 1 df,   p=<2e-16
## Wald test            = 161.9  on 1 df,   p=<2e-16
## Score (logrank) test = 162.2  on 1 df,   p=<2e-16
```

```
####get output of validation data
validate_output=SCGP_get_output(validate_data,SCGP_res)
summary(coxph(Surv(validate_clin$time,validate_clin$status)~validate_output))
```

```
## Call:
## coxph(formula = Surv(validate_clin$time, validate_clin$status) ~
##     validate_output)
##
##   n= 381, number of events= 216
##
##                    coef exp(coef) se(coef)      z Pr(>|z|)
## validate_output -1.4519    0.2341   0.2419 -6.001 1.96e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##                 exp(coef) exp(-coef) lower .95 upper .95
## validate_output    0.2341      4.271    0.1457    0.3762
##
## Concordance= 0.631  (se = 0.022 )
## Likelihood ratio test= 40.74  on 1 df,   p=2e-10
## Wald test            = 36.02  on 1 df,   p=2e-09
## Score (logrank) test = 36.51  on 1 df,   p=2e-09
```

Calculation of importance score

```
library(pracma)
```

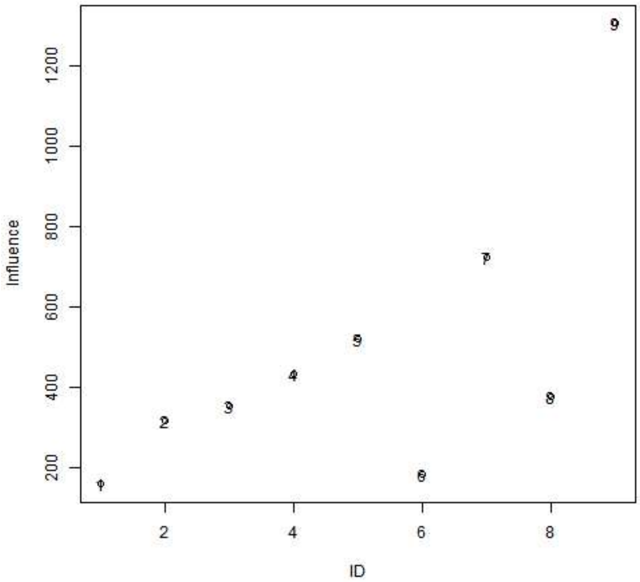
```
##
## 载入程辑包：'pracma'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     sigmoid
```

```
weight=SCGP_res$weight
bias=SCGP_res$bias
imp_occlusion=SCGP_occlusion(data, weight, bias) ##occlusion scores
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

```
###visualization
plot(imp_occlusion)
text(x=imp_occlusion$ID,y=imp_occlusion$Influence,imp_occlusion$ID)
```



Building SCM from pre-defined gene interaction matrix

```
GIM=matrix(sample(c(0,1),size=ncol(data)*ncol(data),replace=TRUE),ncol(data),ncol(data)) ####pre-defined gene interaction matrix
option=list(c(5),c(ncol(GIM)))
SCM.auto=SCGP_autoencoder(GIM,option,learning_rate=0.00001,training_n=50000,message_n=1000)
```

```
## [1] "The iteration is 1000"
## [1] "The error value is 0.769588587125381"
## [1] "The iteration is 2000"
## [1] "The error value is 0.636600558459176"
## [1] "The iteration is 3000"
## [1] "The error value is 0.575620688239302"
## [1] "The iteration is 4000"
## [1] "The error value is 0.540830938525731"
## [1] "The iteration is 5000"
## [1] "The error value is 0.517042597323336"
## [1] "The iteration is 6000"
## [1] "The error value is 0.503905153752171"
## [1] "The iteration is 7000"
## [1] "The error value is 0.496082034346671"
## [1] "The iteration is 8000"
## [1] "The error value is 0.490556832790365"
## [1] "The iteration is 9000"
## [1] "The error value is 0.485695419229107"
## [1] "The iteration is 10000"
## [1] "The error value is 0.48190068037025"
## [1] "The iteration is 11000"
## [1] "The error value is 0.478482543267618"
## [1] "The iteration is 12000"
## [1] "The error value is 0.47528375791346"
## [1] "The iteration is 13000"
## [1] "The error value is 0.472295348465907"
## [1] "The iteration is 14000"
## [1] "The error value is 0.469972567772891"
## [1] "The iteration is 15000"
## [1] "The error value is 0.469997783530784"
## [1] "The iteration is 16000"
## [1] "The error value is 0.471626113446192"
## [1] "The iteration is 17000"
## [1] "The error value is 0.473322654648539"
## [1] "The iteration is 18000"
## [1] "The error value is 0.47514708425831"
## [1] "The iteration is 19000"
## [1] "The error value is 0.477273136328939"
## [1] "The iteration is 20000"
## [1] "The error value is 0.479329287838111"
## [1] "The iteration is 21000"
## [1] "The error value is 0.481703509056364"
## [1] "The iteration is 22000"
## [1] "The error value is 0.483954066552019"
## [1] "The iteration is 23000"
## [1] "The error value is 0.486082836395286"
## [1] "The iteration is 24000"
## [1] "The error value is 0.488328113090108"
## [1] "The iteration is 25000"
## [1] "The error value is 0.490472604645168"
## [1] "The iteration is 26000"
## [1] "The error value is 0.4926969999994"
## [1] "The iteration is 27000"
## [1] "The error value is 0.494830967056237"
## [1] "The iteration is 28000"
## [1] "The error value is 0.496828825068838"
## [1] "The iteration is 29000"
## [1] "The error value is 0.498609097607029"
## [1] "The iteration is 30000"
## [1] "The error value is 0.500214768160599"
```

```
## [1] "The iteration is 31000"
## [1] "The error value is 0.501993105108212"
## [1] "The iteration is 32000"
## [1] "The error value is 0.503549805204285"
## [1] "The iteration is 33000"
## [1] "The error value is 0.504873351485587"
## [1] "The iteration is 34000"
## [1] "The error value is 0.505952969140907"
## [1] "The iteration is 35000"
## [1] "The error value is 0.506779374772847"
## [1] "The iteration is 36000"
## [1] "The error value is 0.507344041084989"
## [1] "The iteration is 37000"
## [1] "The error value is 0.508185749015136"
## [1] "The iteration is 38000"
## [1] "The error value is 0.509131593810931"
## [1] "The iteration is 39000"
## [1] "The error value is 0.509856185522634"
## [1] "The iteration is 40000"
## [1] "The error value is 0.51030440594453"
## [1] "The iteration is 41000"
## [1] "The error value is 0.510480963261175"
## [1] "The iteration is 42000"
## [1] "The error value is 0.510392876442777"
## [1] "The iteration is 43000"
## [1] "The error value is 0.510125810522243"
## [1] "The iteration is 44000"
## [1] "The error value is 0.510211980297198"
## [1] "The iteration is 45000"
## [1] "The error value is 0.510084885400717"
## [1] "The iteration is 46000"
## [1] "The error value is 0.509783054980871"
## [1] "The iteration is 47000"
## [1] "The error value is 0.509689603759741"
## [1] "The iteration is 48000"
## [1] "The error value is 0.509407658526887"
## [1] "The iteration is 49000"
## [1] "The error value is 0.509136234164795"
## [1] "The iteration is 50000"
## [1] "The error value is 0.509414360256363"
```

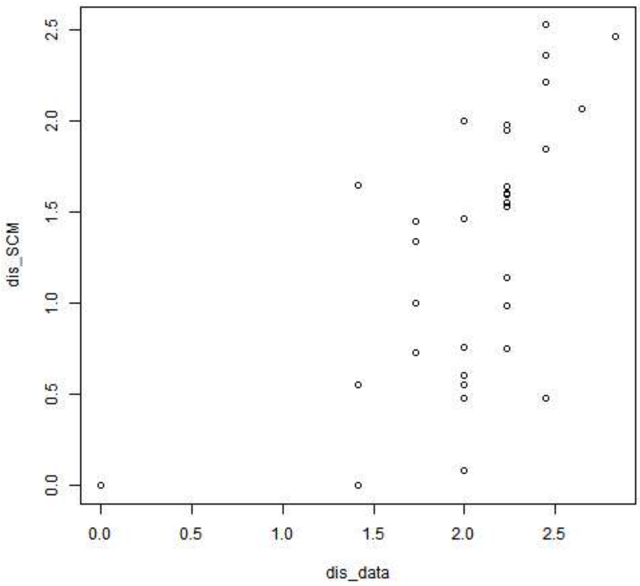
```
SCM.a=SCGP_get_autoencoder(GIM,SCM.auto)
dis_data=melt(distance(GIM, method = "euclidean",test.na = F,use.row.names=T))[,3]
```

```
## Metric: 'euclidean'; comparing: 9 vectors.
```

```
dis_SCM=melt(distance(SCM.a, method = "euclidean",test.na = F,use.row.names=T))[,3]
```

```
## Metric: 'euclidean'; comparing: 9 vectors.
```

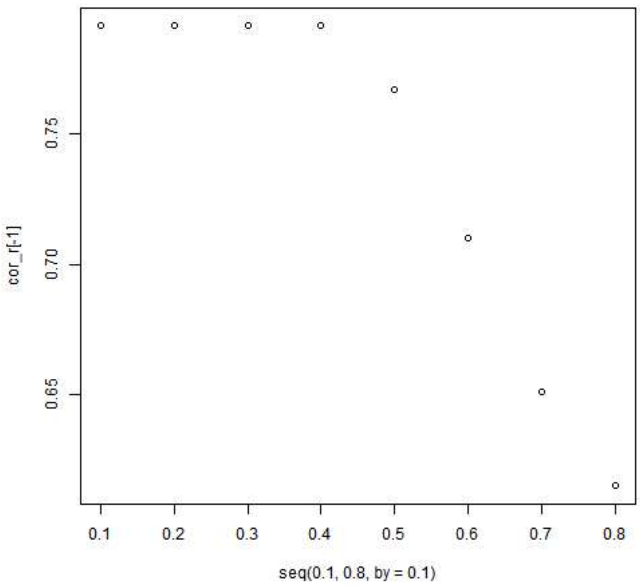
```
plot(dis_data,dis_SCM)
```



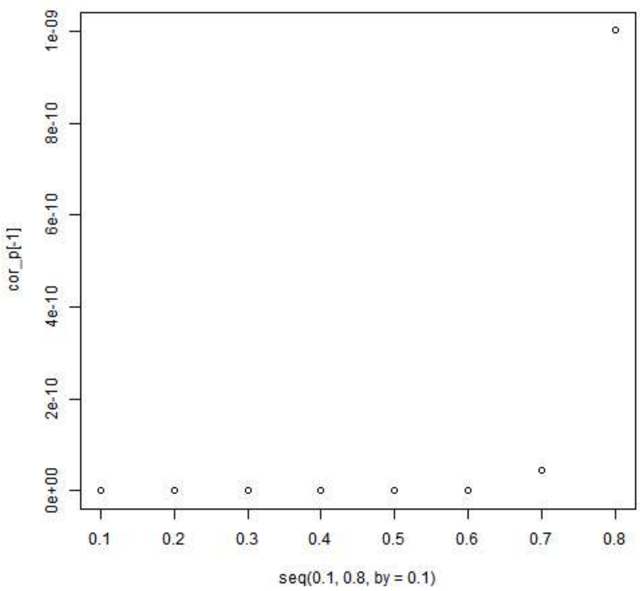

```
cor_r=NA
cor_p=NA
#Por is sparsity
for (Por in seq(0.1,0.8,by=0.1)){
weight_selective=SCM.a
weight_selective[(weight_selective)>quantile(melt(basis(SCM.r))[,3],Por) ]=1
weight_selective[(weight_selective)<=quantile(melt(basis(SCM.r))[,3],Por) ]=0
a2=melt(distance(weight_selective, method = "euclidean",test.na = F,use.row.names=T))[,3]
COR=cor.test(dis_data,a2)
cor_r=c(cor_r,COR$estimate)
cor_p=c(cor_p,COR$p.value)
}
```

```
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
## Metric: 'euclidean'; comparing: 9 vectors.
```

```
plot(seq(0.1,0.8,by=0.1),cor_r[-1])
```




```
plot(seq(0.1,0.8,by=0.1),cor_p[-1])
```




```
Por=0.6
weight_selective=SCM.a
weight_selective[(weight_selective)>quantile(melt(basis(SCM.r))[,3],Por) ]=1
weight_selective[(weight_selective)<=quantile(melt(basis(SCM.r))[,3],Por) ]=0
####
```