Sixteenth International Conference
Financial and Actuarial Mathematics – FAM-2023

ST, DST and ST-based neurons in neural networks

Pavel Stoynov
pavel_todorov_stoynov@yahoo.com

Abstract.
In this article some neurons using ST activation function and similar to them are considered. The article is further development of the research in Stoynov (2016), Stoynov (2021) and stoynov (2022).

Keywords: ST activation function, DST01 neurons, DST01LU neurons.

Introduction

The most natural way to introduce nonlinearity into neural networks is through the nonlinear activation function of the neurons' output.

An overview of the different activation functions used in neurons is for example done by Szandala (2019).

Among the most commonly used are ReLU (Rectified Linear Unit), sigmoid, hyperbolic tangent, Swish (Ramachandran et al., 2017, 2017a) and others.

If is the internal output of the neuron (the output before applying the nonlinearity), then the ReLU function has the form

$$L(n_{kt}) = \max(0, n_{kt}) = n_{kt}^{+}.$$

The sigmoid function has the form

$$L(n_{kt}) = \frac{1}{1+e^{-n_{kt}}}.$$

The hyperbolic tangent function is defined by the equality

$$T(n_{kt}) = \frac{e^{n_{kt}} - e^{-n_{kt}}}{e^{n_{kt}} + e^{-n_{kt}}}.$$

The Swish function has the form

$$L(n_{kt}) = \frac{n_{kt}}{1+e^{-n_{kt}}}.$$

The cumulative Gaussian function is given by the equality

$$\Phi(n_{kt}) = \int_{-\infty}^{n_{kt}} \frac{1}{\sqrt{2\pi}} e^{-0,5x^2}\, dx.$$

The author's proposal is to use the so-called ST activation function, which uses the density of the probability distribution of switches (Switch-Time – ST).

A random variable ξ with a switching distribution ST(n,ß) has a density

$$f_{\xi}(x) = \begin{cases} C(n,\beta) e^{-\beta x} (1+x)^n, & x \geq 0 \\ 0, & x < 0, \end{cases}$$

where $C(n,\beta)$ are normalization coefficients for which

$$C(n,\beta) = \frac{1}{I(n,\beta)}$$

and

$$I(n,\beta)=\frac{1}{\Gamma(1)}\int_0^{\infty}e^{-\beta t}(1+t)^n\,dt=\int_0^{\infty}e^{-\beta t}(1+t)^n\,dt.$$

In private cases when $n=0,\ \ n=1$ and $n=2$ the equalities $C(0,\beta)=\beta,\ \ C(1,\beta)=\frac{\beta^2}{\beta+1}$ and $C(2,\beta)=\frac{\beta^3}{\beta^2+2\beta+2}$ apply accordingly.

Some specific special cases of the distribution are:

1. Exponential distribution: $ST(0,\beta)\equiv Exp(\beta).$
2. Lindley distribution: $ST(1,\beta)\equiv L(\beta).$ A random variable ξ with Lindley distribution has density

$$f_{\xi}(x)=\begin{cases}\frac{\beta^2}{\beta+1}e^{-\beta x}(1+x),\ x\geq 0\\ 0,x<0.\end{cases}$$

3. The distribution $ST(2,\beta).$ The random variable ξ with distribution $ST(2,\beta)$ has density

$$f_{\xi}(x)=\begin{cases}\frac{\beta^3}{\beta^2+2\beta+2}e^{-\beta x}(1+x)^2,\ x\geq 0\\ 0,x<0.\end{cases}$$

Using the ST distribution thus represented, an ST activation function with a threshold of zero can be introduced where

$$N_{kt}=\begin{cases}\int_0^{n_{kt}}C(n,b)e^{-bt}(1+t)^n\,dt,\ n_{kt}\geq 0,\\ 0,\ n_{kt}<0.\end{cases}$$

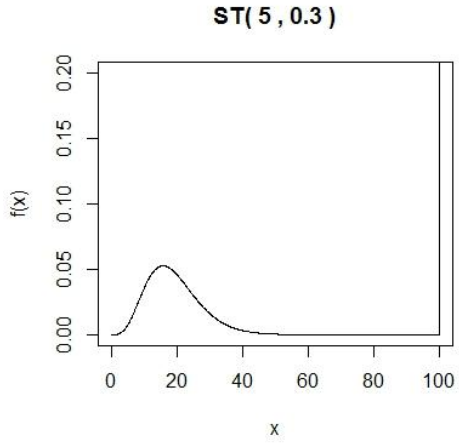

Figure 1. Density of ST distribution with parameters $ST(n=5,\beta=0.3).$

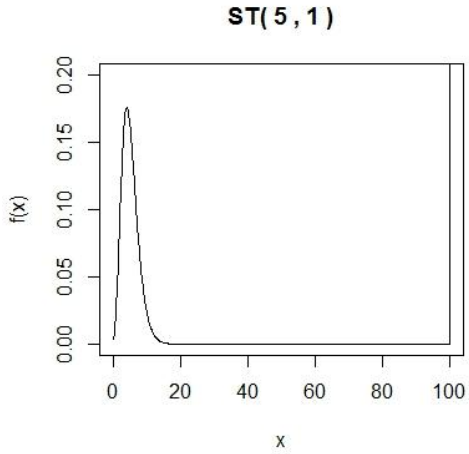

Figure 2. Density of ST distribution with parameters $ST(n=5,\beta=1).$

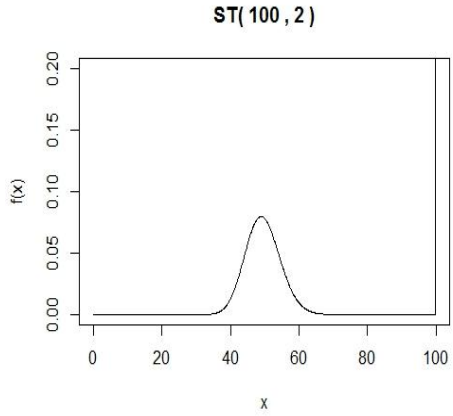

Figure 3. Density of ST distribution with parameters $ST(n=100,\beta=2).$

This function depends on two parameters - parameter n and parameter $b.$ Standard ST activation functions can be obtained by certain selection of these two parameters.

At $n=0$ and $b=1$ a standard exponential activation function is obtained, which has a threshold of zero:

$$N_{kt}=\begin{cases}\int_0^{n_{kt}} e^{-t}dt=1-e^{-n_{kt}},\ n_{kt}\geq 0,\\ 0,\ n_{kt}<0.\end{cases}$$

The graph of this function is presented in Figure 4. At infinity the graph tends asymptotically to unity.

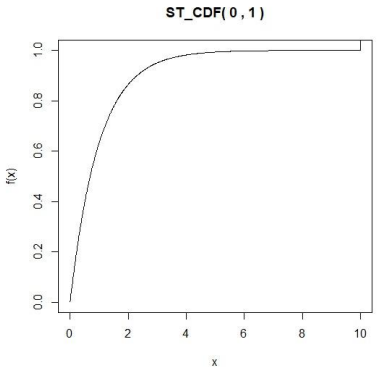

Figure 4. Plot of ST activation function with zero threshold
and parameters $n=0$ and $b=1.$

At $n=1$ and $b=1$ we get a standard Lindley activation function, again with zero threshold:

$$N_{kt}=\begin{cases}\int_0^{n_{kt}}\frac{1}{2}e^{-t}(1+t)dt=1-e^{-n_{kt}}-\frac{1}{2}n_{kt}e^{-n_{kt}},\ n_{kt}\geq 0,\\ 0,\ n_{kt}<0.\end{cases}$$

The graph of this function is presented in Figure 5. At infinity the graph tends asymptotically to unity.

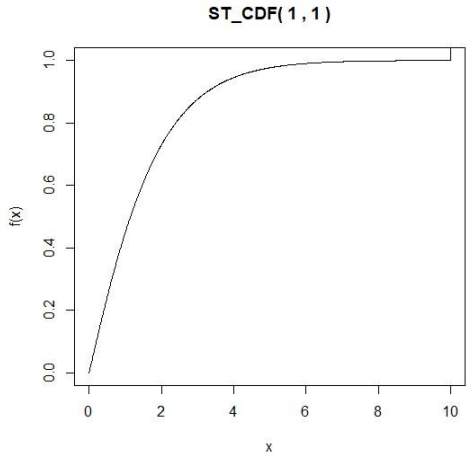

Figure 5. Plot of ST activation function with zero threshold
and parameters $n=1$ and $b=1.$

At $n=2$ and $b=1$ we get an ST(2,1) activation function, again with threshold zero:

$$N_{kt}=\begin{cases}\int_0^{n_{kt}}\frac{1}{5}e^{-t}(1+t)^2dt=1-e^{-n_{kt}}-\frac{4}{5}n_{kt}e^{-n_{kt}}-\frac{1}{5}n_{kt}^2e^{-n_{kt}},\ n_{kt}\geq 0,\\ 0,\ n_{kt}<0.\end{cases}$$

The graph of this function is presented in Figure 6. At infinity the graph tends asymptotically to unity.

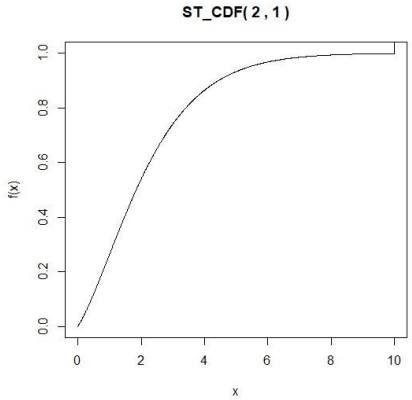

Figure 6. Plot of ST activation function with zero threshold
and parameters $n=2$ and $b=1.$

The double ST distribution (DST distribution) has the density function

$$s_{\xi}(x)=\begin{cases}\frac{1}{2}C(n,b)e^{-bx}(1+x)^{n},\ x\geq 0,\\ \frac{1}{2}C(n,b)e^{bx}(1-x)^{n},x<0.\end{cases}$$

The graph of this function at $n=0$ and $b=1$ is presented in Figure 7.

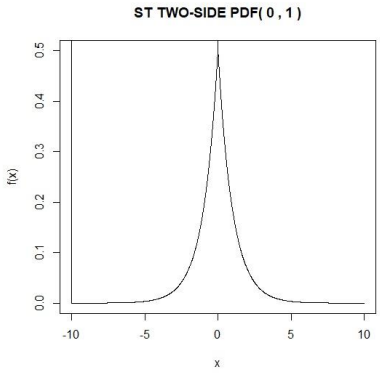

Figure 7. Density plot of double ST distribution with parameters $n=0$ and $b=1.$

The graph of double ST-density at $n=1$ and $b=1$ is presented in Figure 8.

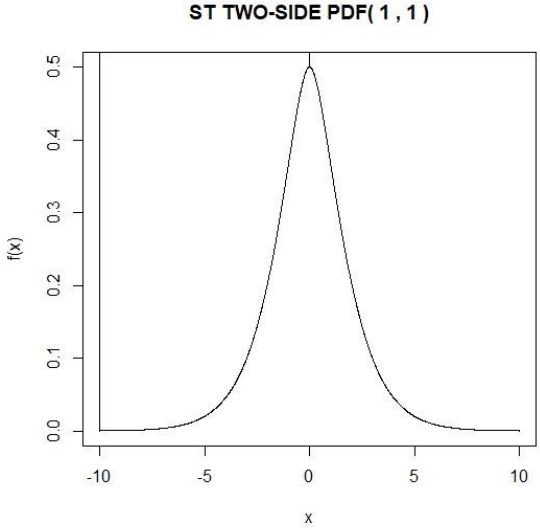

Figure 8. Density plot of double ST distribution with parameters $n=1$ and $b=1.$

The graph of double ST-density at $n=2$ and $b=1$ is presented in Figure 9.

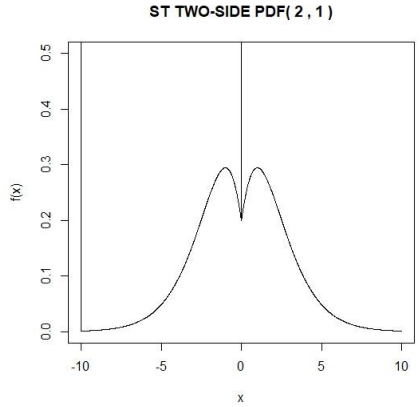

Figure 9. Density plot of double ST distribution
with parameters $n = 2$ and $b = 1.$

This distribution makes it possible to define non-threshold activation functions. The general appearance of the non-threshold ST activation function is:

$$N_{kt} = \int_{-\infty}^{n_{kt}} s(x)dx = \begin{cases} \frac{1}{2} + \frac{1}{2}\int_{0}^{n_{kt}} C(n,b)e^{-bx}(1+x)^n dx, \; n_{kt} \geq 0, \\ \frac{1}{2}\int_{-\infty}^{n_{kt}} C(n,b)e^{bx}(1-x)^n dx, \; n_{kt} < 0. \end{cases}.$$

This function depends on two parameters - the parameter n and $b.$ The standard non-threshold ST activation functions can be obtained by certain selection of these two parameters.

At $n = 0$ and $b = 1$ we get the standard double-exponential activation function, also called the Laplacian activation function:

$$N_{kt} = \begin{cases} \frac{1}{2} + \frac{1}{2}\int_{0}^{n_{kt}} e^{-x}dx = 1 - \frac{1}{2}e^{-n_{kt}}, \; n_{kt} \geq 0, \\ \frac{1}{2}\int_{-\infty}^{n_{kt}} e^{x}dx = \frac{1}{2}e^{n_{kt}}, n_{kt} < 0. \end{cases}$$

At $n = 1$ and $b = 1$ we get a standard ST(1,1) Lindley activation function:

$$N_{kt} = \begin{cases} \frac{1}{2} + \frac{1}{2}\int_{0}^{n_{kt}} \frac{1}{2}e^{-t}(1+t)dt = 1 - \frac{1}{2}e^{-n_{kt}} - \frac{1}{4}n_{kt}e^{-n_{kt}}, \; n_{kt} \geq 0, \\ \frac{1}{2}\int_{-\infty}^{n_{kt}} \frac{1}{2}e^{x}(1-x)dx = \frac{1}{2}e^{n_{kt}} - \frac{1}{4}n_{kt}e^{n_{kt}}, \; n_{kt} < 0. \end{cases}$$

At $n = 2$ and $b = 1$ the ST(2,1) activation function is obtained:

$$N_{kt} = \begin{cases} \frac{1}{2} + \frac{1}{2}\int_{0}^{n_{kt}} \frac{1}{5}e^{-x}(1+x)^2 dx = 1 - \frac{1}{2}e^{-n_{kt}} - \frac{2}{5}n_{kt}e^{-n_{kt}} - \frac{1}{10}n_{kt}^2 e^{-n_{kt}}, \; n_{kt} \geq 0, \\ \frac{1}{2}\int_{-\infty}^{n_{kt}} \frac{1}{5}e^{x}(1-x)^2 dx = \frac{1}{2}e^{n_{kt}} - \frac{2}{5}n_{kt}e^{n_{kt}} + \frac{1}{10}n_{kt}^2 e^{n_{kt}}, \; n_{kt} < 0. \end{cases}$$

The neurons using ST and DST activation function we call ST and DST neurons correspondingly. The neurons using activation functions combining ST and DST with other activation function we call ST-based neurons.

In some of the types of ST neurons, the speed of convergence of the activation function is lower, and therefore a larger number of iterations is needed when training the neural networks.

Therefore, when a higher convergence rate is sought, ST neurons are more useful in combination with neurons with other activation functions, such that ST neurons can be used in the output layers of the network, while the hidden layers can use, for example, a hyperbolic tangent as activation function.

Another possibility is to combine an activation function of type ST with popular types of activation functions such as RELU. Thus, the activation function of a DST01LU neuron is set by the equality

$$N_{kt} = \begin{cases} n_{kt}, n_{kt} \geq 0, \\ \frac{1}{2}\int_{-\infty}^{n_{kt}} e^x dx - \frac{1}{2} = \frac{1}{2}e^{n_{kt}} - \frac{1}{2}, n_{kt} < 0. \end{cases}$$

DST01LU neurons and similar to them are interesting alternatives of the standard conventional neurons used today in neural networks.

Conclusion

Neurons of type ST, DST and ST-based are interesting alternatives of the nurons using the standard popular types of activataion functions in neural networks.

Further research on thir characteristics and possible applications should be performed as following step of studiing these types of neurons.

References

Ramachandran, P., B. Zoph, V. Quoc (2017) Searching for activation functions. CoRR, arXiv, vol. abs/1710.05941. https://arxiv.org/pdf/1710.05941.pdf

Ramachandran, P., B. Zoph, V. Quoc (2017a) SWISH: a self-gated activation function. CoRR, arXiv, https://arxiv.org/pdf/1710.05941v1.pdf?source=post_page

Stoynov, P. (2016) Switch Time Family of distributions and processes and their applications to reflected surplus models. - Annual of the Faculty of Economics and Business Administration, Sofia University "St. Kliment Ohridski" - Sofia, 2016, 255-285.

Stoynov, P. (2021) Applications of ST Distributions to Neural Networks and Regression Models. Proceedings of Eighth International Conference on New Trends in the Applications of Differential Equations in Science (NTADES'21), 6-9 September, St. Constantine and Helena, Bulgaria.

Stoynov, P. (2022) Switch Time Activation Function and Stopit Regression – Some Examples - Proceedings of XXXI International Scientific Conference Electronics-ET2022, 13-15 September 2022, Sozopol, Bulgaria.

Szandala, T. (2019) Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. arXiv. Cornell University. https://arxiv.org/ftp/arxiv/papers/2010/2010.09458.pdf

Appendices

Appendix 1. Code in R related to ST neurons

```
#ST(n,beta) graphics
beta<-0.3
n<-5
In_beta<-function(n,beta, recursive=TRUE)
{if (n==0) return (1/beta) else return (1/beta+n/beta*In_beta(n-1,beta))}
Cn_beta<-function(n,beta) {1/In_beta(n,beta)}
STn_beta<-function(x,n,beta){Cn_beta(n,beta)*(1+x)^n*exp(-x*beta)}
m<-10000
STn_beta_values<-0:m
T<-100
tvec<-T*0:m/m
for (i in 1:m) {
```

```
      STn_beta_values[i]<-STn_beta(i*T/m,n,beta)
}
freq<-10   # take out every 1000th point in the simulation & plot it
pick<-freq*(0:(m/freq))+1
title<-paste("ST(", round(n,1), ",",round(beta,1), ")")
plot(tvec[pick],STn_beta_values[pick],type='l',xlab="x",ylab="f(x)",main=title,ylim=c(0,0.2))

#D2 simulation
n<-2
m<-100
Pr<-0
beta<-3
Dn_values<-1:(n+1)
#Dn_prob<-(1:m,1:(n+1))
Dn_prob <- matrix(1:m*(n+1), ncol = (n+1), nrow = m)
for (i in 1:m) {
      beta<-i
      Dn_prob[i,1]<-beta^2/(beta^2+2*beta+2)
      Dn_prob[i,2]<- 2*beta/(beta^2+2*beta+2)
      Dn_prob[i,3]<-2 /(beta^2+2*beta+2)
}
Dn_prob


#Dn graphics
n<-6
m<-100
beta<-0.1
Dn_values<-1:(n+1)
Dn_prob<-1:(n+1)
In_beta<-function(n,beta, recursive=TRUE)
{if (n==0) return (1/beta) else return (1/beta+n/beta*In_beta(n-1,beta))}
Cn_beta<-function(n,beta) {1/In_beta(n,beta)}
for (i in 1:(n+1)){Dn_prob[i]<-(Cn_beta(n,beta)*factorial(n)/(beta^i*factorial(n-i+1)))}
title<-paste("Dn graphics for n=",round(n,1), " and beta=", round(beta,2))
plot(Dn_prob, main=title)

#EDn
In_beta<-function(n,beta, recursive=TRUE)
{if (n==0) return (1/beta) else return (1/beta+n/beta*In_beta(n-1,beta))}
Cn_beta<-function(n,beta) {1/In_beta(n,beta)}
EDn<-function(n,beta)
{
Dn_values<-1:(n+1)
Dn_prob<-1:(n+1)
for (i in 1:(n+1))
Dn_prob[i]<-(Cn_beta(n,beta)*factorial(n)/(beta^i*factorial(n-i+1)))
expectation<-0
for (k in 1:(n+1))
{
a_value<-Dn_prob[k]*Dn_values[k]
expectation<-(expectation+a_value)
}
return (expectation)
}
exp<-EDn(5,0.3)
exp


#EST(n,beta)
```

```
beta<-0.3
n<-5
In_beta<-function(n,beta, recursive=TRUE)
{if (n==0) return (1/beta) else return (1/beta+n/beta*In_beta(n-1,beta))}
Cn_beta<-function(n,beta) {1/In_beta(n,beta)}
STn_beta<-function(x,n,beta){Cn_beta(n,beta)*(1+x)^n*exp(-x*beta)}
EST<- function(n,beta){
expectation<-0
for (i in 0:n) {
      expectation<- expectation+Cn_beta(n,beta)*In_beta(i+1, beta, TRUE)* factorial(n)/ (factorial(i)
*factorial(n-i))
}
return (expectation)
}
EST(n,beta)

#ST(n,beta) graphics
beta<-0.3
n<-5

In_beta<-function(n,beta, recursive=TRUE)
{if (n==0) return (1/beta) else return (1/beta+n/beta*In_beta(n-1,beta))}

Cn_beta<-function(n,beta) {1/In_beta(n,beta)}

STn_beta<-function(x,n,beta){Cn_beta(n,beta)*(1+x)^n*exp(-x*beta)}

m<-10000
STn_beta_values<-0:m
T<-100
tvec<-T*0:m/m
for (i in 1:m) {
      STn_beta_values[i]<-STn_beta(i*T/m,n,beta) }
freq<-10   # take out every 1000th point in the simulation & plot it
pick<-freq*(0:(m/freq))+1
title<-paste("ST(", round(n,1), ",",round(beta,1), ")")
plot(tvec[pick],STn_beta_values[pick],type='l',xlab="x",ylab="f(x)",main=title,ylim=c(0,0.2))


N<-10
ST_n<-0
ST_beta<-1
ST_simulated<-1:N
for (i in 1:N)
{
ST_simulated[i]<-rgamma(1,ST_n+1,1/ST_beta)
}

#ST01 activation function with scalar input and output
ST01<-function(x){
{if (x<=0) return (0) else return (1-exp(-x))}
}

#ST01_neuron
ST01_neuron<-function(b,x){
z<-0
m<-length(x)
for (i in 1:m)
{
z=(z+x[i]*b[i])
```

```
}
return (ST01(z))
}


#ST11 activation function with scalar input and output
ST11<-function(x){
{if (x<=0) return (0)
else
return (1-exp(-x)-1/2*x*exp(-x))}
}

#ST11_neuron
ST11_neuron<-function(b,x){
z<-0
m<-length(x)
for (i in 1:m)
{
z=(z+x[i]*b[i])
}
return (ST11(z))
}



#ST21 activation function with scalar input and output
ST21<-function(x){
{if (x<=0) return (0)
else
return (1-exp(-x)-4/5*x*exp(-x)-1/5*x^2*exp(-x))}
}

#ST21_neuron
ST21_neuron<-function(b,x){
z<-0
m<-length(x)
for (i in 1:m)
{
z=(z+x[i]*b[i])
}
return (ST21(z))
}


#DST01 activation function with scalar input and output
DST01<- function(x) {
  if (x>=0)
   return(1-1/2*exp(-x))
else
return(1/2*exp(x))
}

#DST01_neuron
DST01_neuron<-function(b,x){
z<-0
m<-length(x)
for (i in 1:m)
{
z=(z+x[i]*b[i])
}
```

```
return (DST01(z))
}

#DST11 activation function with scalar input and output
DST11<- function(x) {
  if (x>=0)
   return(1-1/2*exp(-x)-1/4*x*exp(-x))
else
   return(1/2*exp(x)-1/4*x*exp(x))
}

#DST11_neuron
ST_neuron_OS_1_1<-function(b,x){
z<-0
m<-length(x)
for (i in 1:m)
{
z=(z+x[i]*b[i])
}
return (DST11(z))
}


#DST21 activation function with scalar input and output
DST21<- function(x) {
  if (x>=0)
   return(1-1/2*exp(-x)-2/5*x*exp(-x)-1/10*x^2*exp(-x))
else
 return(1/2*exp(x)-2/5*x*exp(x)+1/10*x^2*exp(x))
}

#DST21_neuron
DST21_neuron<-function(b,x){
z<-0
m<-length(x)
for (i in 1:m)
{
z=(z+x[i]*b[i])
}
return (DST21(z))
}
```

Appendix 2. Code in Python related to ST neurons

```
def ST01(x):
   if x<0:
            return 0.0
   else:
            return(1-math.exp(-x))

def ST01_neuron(b,x):
   z=0
   m=len(x)
   for i in range(m):
```

```
            z=(z+x[i]*b[i])
        return ST01(z)

def ST11(x):
        if x<0:
                return 0.0
        else:
                return(1-math.exp(-x)-1/2*x*math.exp(-x))

def ST11_neuron(b,x):
        z=0
        m=len(x)
        for i in range(m):
       z=(z+x[i]*b[i])
        return ST11(z)

def ST21(x):
        if x<0:
                return 0.0
        else:
                return(1-math.exp(-x)-4/5*x*math.exp(-x)-1/5*x**2*math.exp(-x))

def ST21_neuron(b,x):
        z=0
        m=len(x)
        for i in range(m):
       z=(z+x[i]*b[i])
        return ST21(z)


      def DST01(x):
        if x>=0:
                return(1-1/2*math.exp(-x))
        else:
                return(1/2*math.exp(x))

      def DST01_neuron(b,x):
        z=0
        m=len(x)
        for i in range(m):
            z=(z+x[i]*b[i])
        return DST01(z)

def DST11(x):
        if x>=0:
                return(1-1/2*math.exp(-x)-1/4*x*math.exp(-x))
        else:
                return(1/2*math.exp(x)-1/4*x*math.exp(x))

def DST11_neuron(b,x):
        z=0
        m=len(x)
        for i in range(m):
       z=(z+x[i]*b[i])
        return DST11(z)


      def DST21(x):
        if x>=0:
                return(1-1/2*math.exp(-x)-2/5*x*math.exp(-x)-1/10*x**2*math.exp(-x))
```

```
        else:
                return(1/2*math.exp(x)-2/5*x*math.exp(x)+1/10*x**2*math.exp(x))

    def DST21_neuron(b,x):
        z=0
        m=len(x)
        for i in range(m):
            z=(z+x[i]*b[i])
        return DST21(z)

#DST01LU activation function with array input and output
                def DST01LU(x):
                output= x
                m=len(x[0])
                for i in range(m):
                    if x[0][i]>=0:
                            output[0][i]=x[0][i]
                    else:
                            output[0][i]=(1/2*math.exp(x[0][i])-1/2)
                return output
#DST01LU neuron
def DST01LU_prime(x):
                output= x
                m=len(x[0])
                for i in range(m):
                    if x[0][i]>=0:
                            output[0][i]=1
                    else:
                            output[0][i]=(1/2*math.exp(x[0][i]))
                return output
```