Matrix

2D Array

eye

```
>>> np.eye(5)
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

ones

小括號代表tuple，
描述陣列大小

```
>>> np.ones((5,3))
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

zeros

```
>>> np.zeros((3,5))
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

diag

使用方括號，代表串列，會成為矩陣的對角元素

```
>>> np.diag([1,2,3,4,5])
array([[1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 4, 0],
       [0, 0, 0, 0, 5]])
```

使用range, reshape 生成二維陣列

```
[[ 1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18]
 [19 20 21 22 23 24 25 26 27]
 [28 29 30 31 32 33 34 35 36]
 [37 38 39 40 41 42 43 44 45]
 [46 47 48 49 50 51 52 53 54]
 [55 56 57 58 59 60 61 62 63]
 [64 65 66 67 68 69 70 71 72]
 [73 74 75 76 77 78 79 80 81]]
```

```
A = np.array([range(1,82)])
B = np.reshape(A,(9,9))
print(B)
```

生成一維陣列，元素包含1到81

將一維陣列，排列為二維陣列

```
[[ 1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18]
 [19 20 21 22 23 24 25 26 27]
 [28 29 30 31 32 33 34 35 36]
 [37 38 39 40 41 42 43 44 45]
 [46 47 48 49 50 51 52 53 54]
 [55 56 57 58 59 60 61 62 63]
 [64 65 66 67 68 69 70 71 72]
 [73 74 75 76 77 78 79 80 81]]
```

使用shape列印陣列大小

```
A = np.array([range(1,82)])
B = np.reshape(A,(9,9))
print(np.shape(A))
print(np.shape(B))
```

(9,9)代表tuple

```
(1, 81)
(9, 9)
```

轉置二維陣列

```
A = np.array([range(1,82)])
B = np.reshape(A,(9,9))
C = np.transpose(B)
print(C)
```

```
[[ 1 10 19 28 37 46 55 64 73]
 [ 2 11 20 29 38 47 56 65 74]
 [ 3 12 21 30 39 48 57 66 75]
 [ 4 13 22 31 40 49 58 67 76]
 [ 5 14 23 32 41 50 59 68 77]
 [ 6 15 24 33 42 51 60 69 78]
 [ 7 16 25 34 43 52 61 70 79]
 [ 8 17 26 35 44 53 62 71 80]
 [ 9 18 27 36 45 54 63 72 81]]
```

array

```
A = np.array([range(1,5)])
```

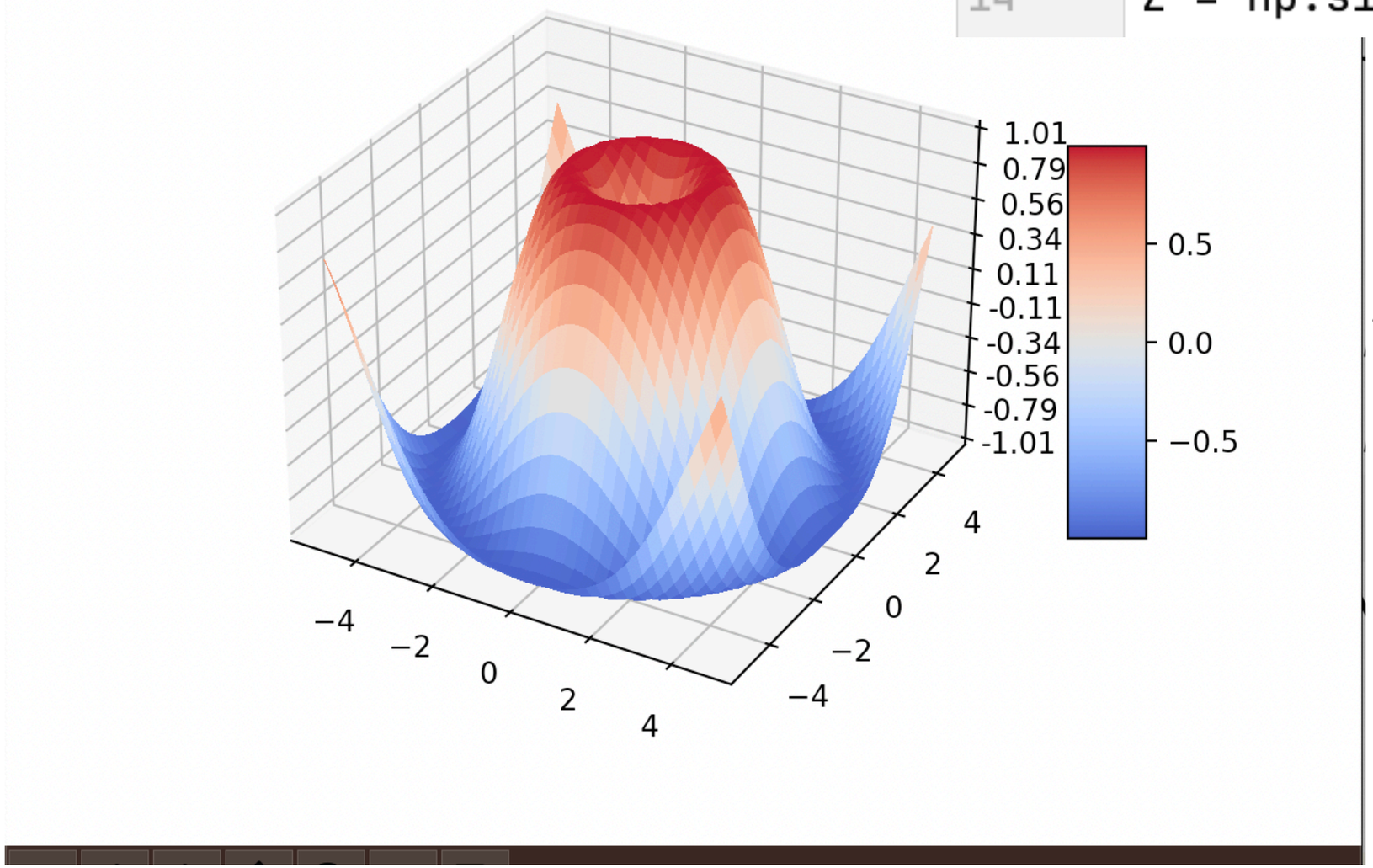
```
>>> A = np.array([range(1,5)])
>>> A
array([[1, 2, 3, 4]])
```

array Transpose

```
A = np.array([range(1,5)])
np.transpose(A)
```

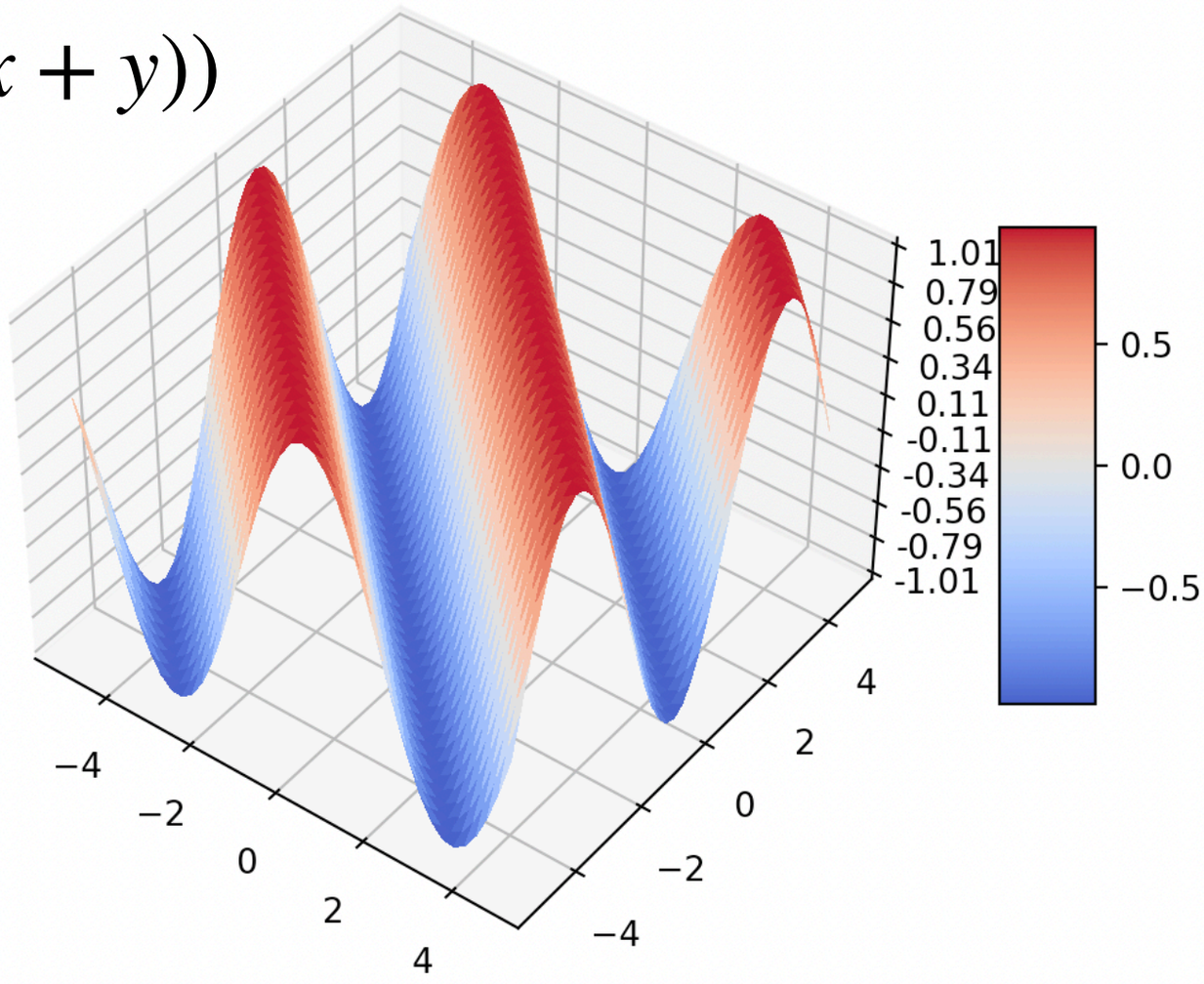
```
>>> A = np.array([range(1,5)])
... B = np.transpose(A)
>>> B
array([[1],
       [2],
       [3],
       [4]])
```

```
10    X = np.arange(-5, 5, 0.25)
11    Y = np.arange(-5, 5, 0.25)
12    X, Y = np.meshgrid( *xi: X, Y)
13    R = np.sqrt(X**2 + Y**2)
14    Z = np.sin(R)
```



$$f(x, y) = sin(\sqrt{(x^2 + y^2)})$$

$$f(x, y) = sin(x + y))$$




```
 9    # Make data.
10    X = np.arange(-5, 5, 0.25)
11    Y = np.arange(-5, 5, 0.25)
12    X, Y = np.meshgrid( *xi: X, Y)
13    Z = np.sin(X+Y)
```

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
from matplotlib.ticker import LinearLocator
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
# Make data.
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
Z = np.sin(X+Y)
# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
# Customize the z axis.
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
# A StrMethodFormatter is used automatically
ax.zaxis.set_major_formatter('{x:.02f}')
# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```

@代表矩陣乘法

```
n = 4
P = np.random.randint(0,10,(n,n))
print(P)
S = P @ P.T
S = np.matmul(P,P.T)
```

P.T 代表P的轉置

```
>>> P
array([[6, 6, 5, 2],
       [9, 0, 3, 8],
       [7, 6, 0, 3],
       [4, 7, 1, 1]])
```

```
>>> P.T
array([[6, 9, 7, 4],
       [6, 0, 6, 7],
       [5, 3, 0, 1],
       [2, 8, 3, 1]])
```

```
>>> S
array([[101,  85,  84,  73],
       [ 85, 154,  87,  47],
       [ 84,  87,  94,  73],
       [ 73,  47,  73,  67]])
```

```
>>> S2
array([[101,  85,  84,  73],
       [ 85, 154,  87,  47],
       [ 84,  87,  94,  73],
       [ 73,  47,  73,  67]])
>>>
```

10.14

START

import numpy as np
initialization: B,C

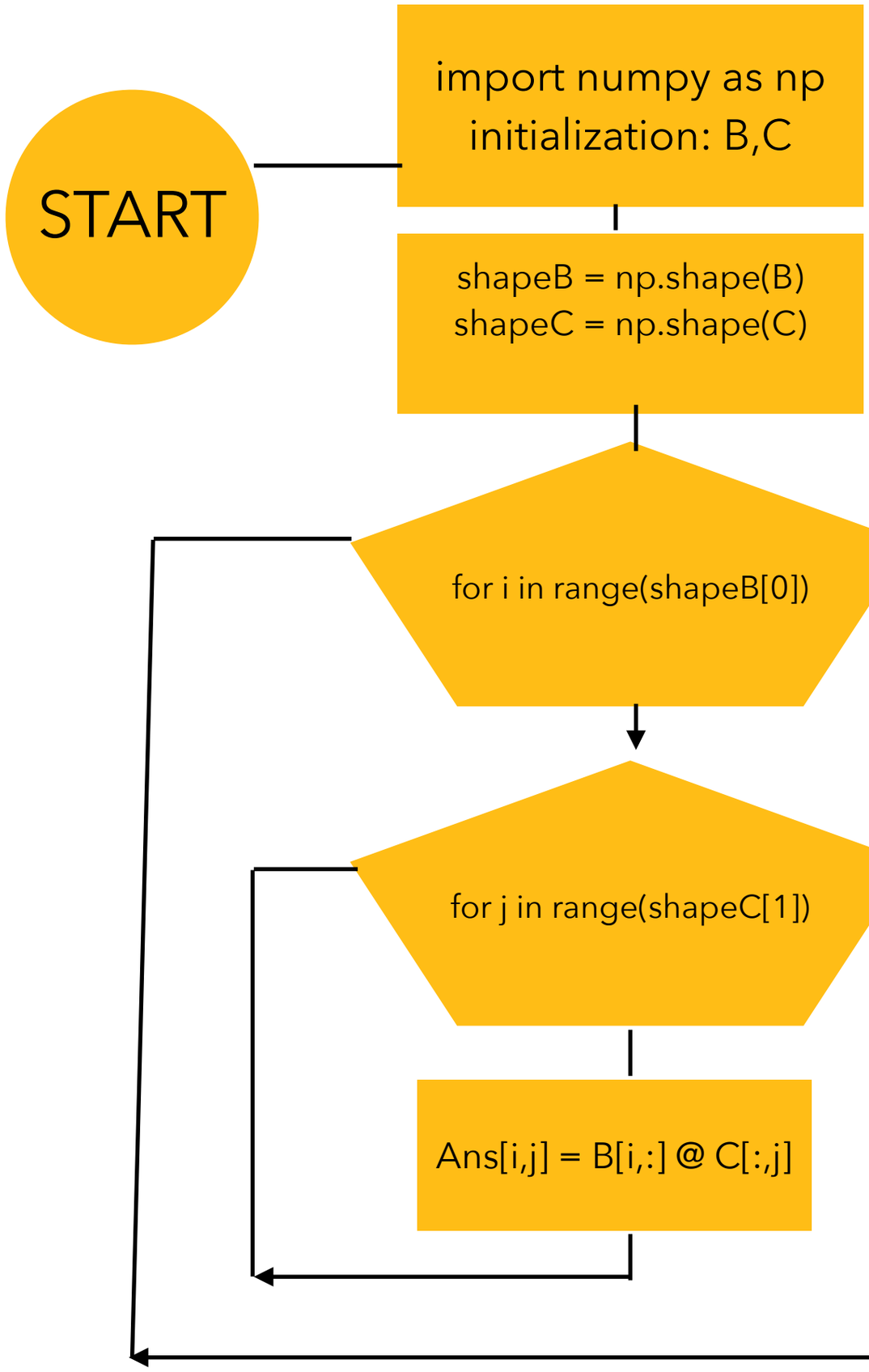
shapeB = np.shape(B)
shapeC = np.shape(C)

for i in range(shapeB[0])

end

for j in range(shapeC[1])

Ans[i,j] = B[i,:] @ C[:,j]




```
[[2 5 3 9]  [[2 3 1 7]
 [3 8 3 0]   [5 8 7 9]
 [1 7 3 0]   [3 3 3 5]
 [7 9 5 0]]  [9 0 0 0]]
```

```
shapeB = np.shape(B)
shapeC = np.shape(C)
for i in range(shapeB[0]):
    for j in range(shapeC[1])
        ans[i,j] = B[i,:] @ C[:,j]
```

```
[[2 5 3 9]
 [3 8 3 0]
 [1 7 3 0]
 [7 9 5 0]]
```

×

```
[[2 3 1 7]
 [5 8 7 9]
 [3 3 3 5]
 [9 0 0 0]]
```

```
[[119  55  46  74]
 [ 55  82  68 108]
 [ 46  68  59  85]
 [ 74 108  85 155]]
```

子陣列

```
[[ 1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18]
 [19 20 21 22 23 24 25 26 27]
 [28 29 30 31 32 33 34 35 36]
 [37 38 39 40 41 42 43 44 45]
 [46 47 48 49 50 51 52 53 54]
 [55 56 57 58 59 60 61 62 63]
 [64 65 66 67 68 69 70 71 72]
 [73 74 75 76 77 78 79 80 81]]
```

```
[[ 1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18]
 [19 20 21 22 23 24 25 26 27]
 [28 29 30 31 32 33 34 35 36]
 [37 38 39 40 41 42 43 44 45]
 [46 47 48 49 50 51 52 53 54]
 [55 56 57 58 59 60 61 62 63]
 [64 65 66 67 68 69 70 71 72]
 [73 74 75 76 77 78 79 80 81]]
```

```
A = np.array([range(1,82)])
B = np.reshape(A,(9,9))
C = B[1:9,0:4]
```

```
array([[10, 11, 12, 13],
       [19, 20, 21, 22],
       [28, 29, 30, 31],
       [37, 38, 39, 40],
       [46, 47, 48, 49],
       [55, 56, 57, 58],
       [64, 65, 66, 67],
       [73, 74, 75, 76]])
```

子陣列

C = B[1:9,5:9]

```
[[ 1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18]
 [19 20 21 22 23 24 25 26 27]
 [28 29 30 31 32 33 34 35 36]
 [37 38 39 40 41 42 43 44 45]
 [46 47 48 49 50 51 52 53 54]
 [55 56 57 58 59 60 61 62 63]
 [64 65 66 67 68 69 70 71 72]
 [73 74 75 76 77 78 79 80 81]]
```

```
[[ 1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18]
 [19 20 21 22 23 24 25 26 27]
 [28 29 30 31 32 33 34 35 36]
 [37 38 39 40 41 42 43 44 45]
 [46 47 48 49 50 51 52 53 54]
 [55 56 57 58 59 60 61 62 63]
 [64 65 66 67 68 69 70 71 72]
 [73 74 75 76 77 78 79 80 81]]
```

```
A = np.array([range(1,82)])
B = np.reshape(A,(9,9))
C = B[1:9,5:9]
```

```
[[15, 16, 17, 18],
 [24, 25, 26, 27],
 [33, 34, 35, 36],
 [42, 43, 44, 45],
 [51, 52, 53, 54],
 [60, 61, 62, 63],
 [69, 70, 71, 72],
 [78, 79, 80, 81]]
```

子陣列

A = np.array([range(1,82)])

B = np.reshape(A,(9,9))

C = B[3:8,2:6]

```
    0  1.  2.  3.  4.  5.  6.  7  8
0  [[ 1   2   3   4   5   6   7   8   9]
1   [10  11  12  13  14  15  16  17  18]
2   [19  20  21  22  23  24  25  26  27]
3   [28  29  30  31  32  33  34  35  36]
4   [37  38  39  40  41  42  43  44  45]
5   [46  47  48  49  50  51  52  53  54]
6   [55  56  57  58  59  60  61  62  63]
7   [64  65  66  67  68  69  70  71  72]
8   [73  74  75  76  77  78  79  80  81]]
```

將兩個矩陣水平堆疊

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

```
A = np.array([range(1,26)])
B = np.reshape(A,(5,5))
C = B[1:5,0:2]
D = B[1:5,3:5]
```

```
[ 6,  7]
[11, 12]
[16, 17]
[21, 22]
```

```
[[ 9, 10],
 [14, 15],
 [19, 20],
 [24, 25]]
```

```
A = np.array([range(1,26)])
B = np.reshape(A,(5,5))
C = B[1:5,0:2]
D = B[1:5,3:5]
E = np.hstack((C,D))
```

用小括號，代表tuple，
其中可以包括多個矩陣

```
array([[ 6,  7,  9, 10],
       [11, 12, 14, 15],
       [16, 17, 19, 20],
       [21, 22, 24, 25]])
```

```
>>> E = np.hstack((C,D,C,D))
>>> E
array([[ 6,  7,  9, 10,  6,  7,  9, 10],
       [11, 12, 14, 15, 11, 12, 14, 15],
       [16, 17, 19, 20, 16, 17, 19, 20],
       [21, 22, 24, 25, 21, 22, 24, 25]])
```

將兩個矩陣垂直堆疊

```
A = np.array([range(1,26)])
B = np.reshape(A,(5,5))
C = B[1:5,0:2]
D = B[1:5,3:5]
E = np.vstack((C,D))
```

```
array([[ 6,  7],
       [11, 12],
       [16, 17],
       [21, 22],
       [ 9, 10],
       [14, 15],
       [19, 20],
       [24, 25]])
```

矩陣相加

```
A = np.array([range(1,26)])
B = np.reshape(A,(5,5))
I = np.eye(5)
B = B + I
```

```
array([[ 2.,  2.,  3.,  4.,  5.],
       [ 6.,  8.,  8.,  9., 10.],
       [11., 12., 14., 14., 15.],
       [16., 17., 18., 20., 20.],
       [21., 22., 23., 24., 26.]])
```

隨機陣列

```
>>> np.random.rand(3,2)
array([[0.93655909, 0.95197349],
       [0.04122792, 0.98316528],
       [0.33106862, 0.37519673]])
```

```
>>> np.random.rand(3,2)*2-1
array([[-0.46935926, -0.58594718],
       [ 0.75553641,  0.68798631],
       [ 0.71628121, -0.08510201]])
```

Inner product to arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([4,5,6])
>>> np.transpose(a) @ b
32
```

Row vector, not a matrix with one row

矩陣相乘

```
shapeB = np.shape(B)
shapeC=np.shape(C)
ans = np.zeros((shapeB[0],shapeC[1]))
for i in range(shapeB[0]):
    for j in range(shapeC[1]):
        ans[i,j] = B[i,:] @ C[:,j]
```

將矩陣的對角元素加1

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

單位矩陣

```
D = np.array([range(1,26)])
I = np.eye(5)
A = np.reshape(D,(5,5))
A = A+I
```

```
[[ 2.,  2.,  3.,  4.,  5.],
 [ 6.,  8.,  8.,  9., 10.],
 [11., 12., 14., 14., 15.],
 [16., 17., 18., 20., 20.],
 [21., 22., 23., 24., 26.]]
```

對角線元素加1

給定A與x，以矩陣乘法計算b

$$Ax = b$$

```
D = np.array([range(1,26)])
I = np.eye(5)
A = np.reshape(D,(5,5))
A = A+I
x = np.array([[1],[0],[1],[-1],[1]])
b = np.matmul(A,x)
```

```
[[ 2.,  2.,  3.,  4.,  5.],   [[ 1],      [[ 6.],
 [ 6.,  8.,  8.,  9., 10.],    [ 0],       [15.],
 [11., 12., 14., 14., 15.],    [ 1],       [26.],
 [16., 17., 18., 20., 20.],    [-1],       [34.],
 [21., 22., 23., 24., 26.]]    [ 1]]       [46.]]
```

$$Ax = b$$

如果A為可逆矩陣

$$x = A^{-1}b$$

給定A與b，以反矩陣運算與矩陣乘法解x

反矩陣運算inv

```
from numpy.linalg import inv
invA = inv(A)
```

```
[[ 0.19021739, -0.5326087 , -0.25543478,  0.02173913,  0.29891304],
 [-0.51086957,  0.63043478, -0.22826087, -0.08695652,  0.05434783],
 [-0.21195652, -0.20652174,  0.79891304, -0.19565217, -0.19021739],
 [ 0.08695652, -0.04347826, -0.17391304,  0.69565217, -0.43478261],
 [ 0.38586957,  0.11956522, -0.14673913, -0.41304348,  0.32065217]]
```

以反矩陣運算解線性系統

```
from numpy.linalg import inv
invA = inv(A)
xhat = invA @ b
```

```
[[ 1.00000000e+00],
 [ 1.77635684e-15],
 [ 1.00000000e+00],
 [-1.00000000e+00],
 [ 1.00000000e+00]]
```

反矩陣

```
n = 5
A = np.random.rand(n,n)
from numpy.linalg import inv
invA = inv(A)
I2 = np.matmul(A,invA)
```

I2為單位矩陣

A 與A的反矩陣相乘，應該會得到單位矩陣

反矩陣

```
n = 5
A = np.random.rand(n,n)
from numpy.linalg import inv
invA = inv(A)
I2 = np.matmul(A,invA)
I = np.eye(n)
absErr = np.sum(np.sum(np.abs(I - I2)))
```

```
>>> absErr
3.207674463751382e-15
```

計算I2與np.eye(n)差的所有元素的絕對值總和

大型矩陣的反矩陣

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,4999} & a_{1,5000} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,4999} & a_{2,5000} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{4999,1} & a_{4999,2} & \cdots & a_{4999,4999} & a_{4999,5000} \\ a_{5000,1} & a_{5000,2} & \cdots & a_{5000,4999} & a_{5000,5000} \end{bmatrix}$$

大型矩陣的反矩陣

隨機產生
5000 × 5000
的矩陣A

```
n = 5000
A = np.random.rand(n,n)
from numpy.linalg import inv
invA = inv(A)
I2 = np.matmul(A,invA)
I = np.eye(n)
absErr = np.sum(np.sum(np.abs(I - I2)))
```

```
>>> absErr
3.207674463751382e-15
```

大型矩陣的反矩陣

計算矩陣A的反矩陣

```
n = 5000
A = np.random.rand(n,n)
from numpy.linalg import inv
invA = inv(A)
I2 = np.matmul(A,invA)
I = np.eye(n)
absErr = np.sum(np.sum(np.abs(I - I2)))
```

```
>>> absErr
3.207674463751382e-15
```

大型矩陣的反矩陣

```
n = 5000
A = np.random.rand(n,n)
from numpy.linalg import inv
invA = inv(A)
I2 = np.matmul(A,invA)
I = np.eye(n)
absErr = np.sum(np.sum(np.abs(I - I2)))
```

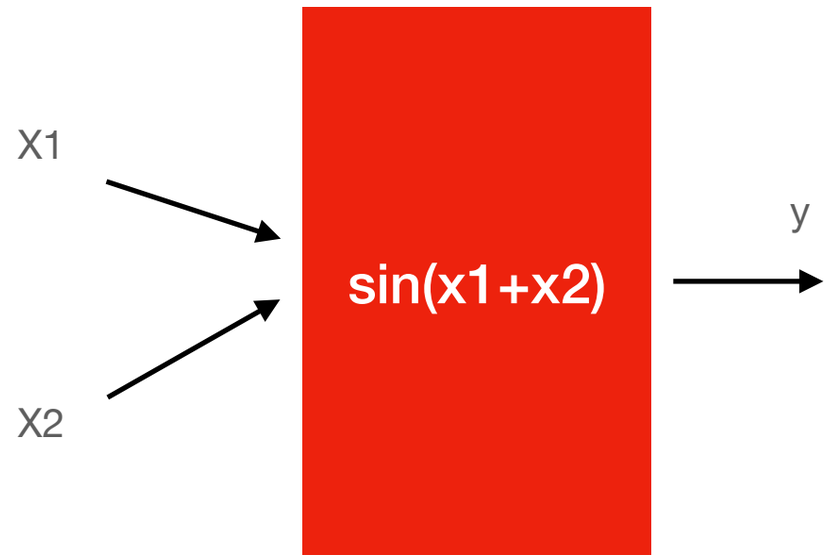
比較AA^{-1}與單位矩陣

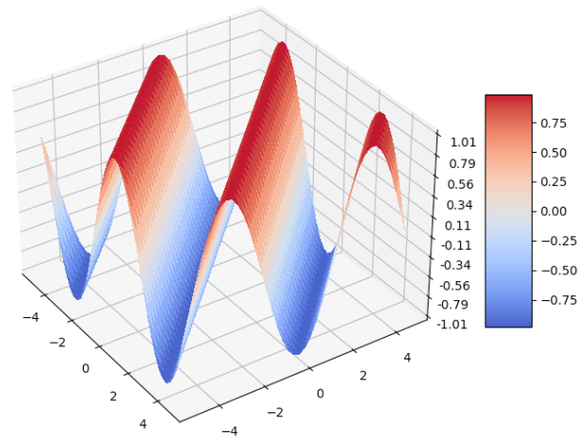
```
>>> absErr
3.207674463751382e-15
```

```
>>> n = 5000
... A = np.random.rand(n,n)
... from numpy.linalg import inv
... invA = inv(A)
... I2 = np.matmul(A,invA)
... I = np.eye(n)
... absErr = np.sum(np.sum(np.abs(I - I2)))
>>> absErr
0.00010767582085438 7
```

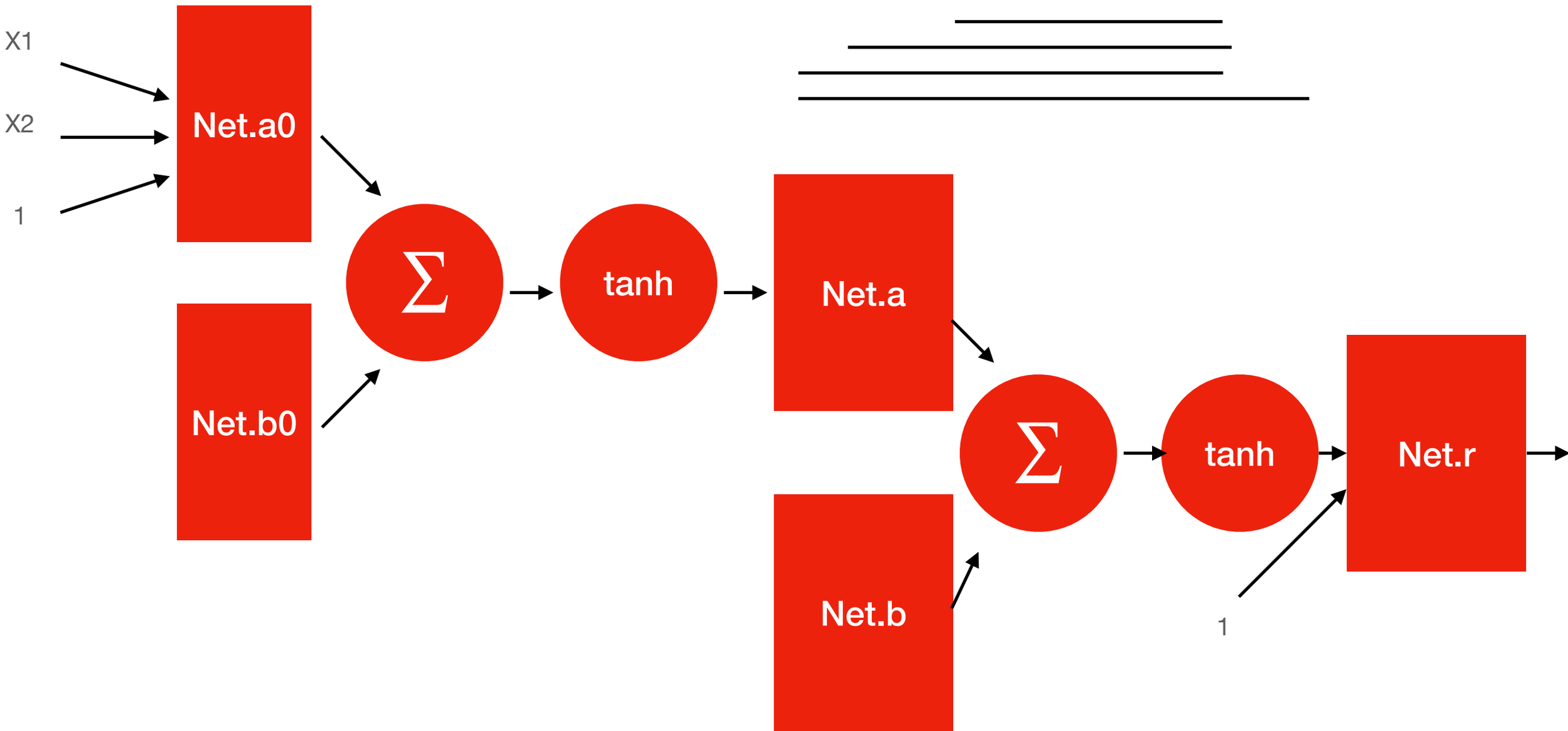
Mean square error 10^-6

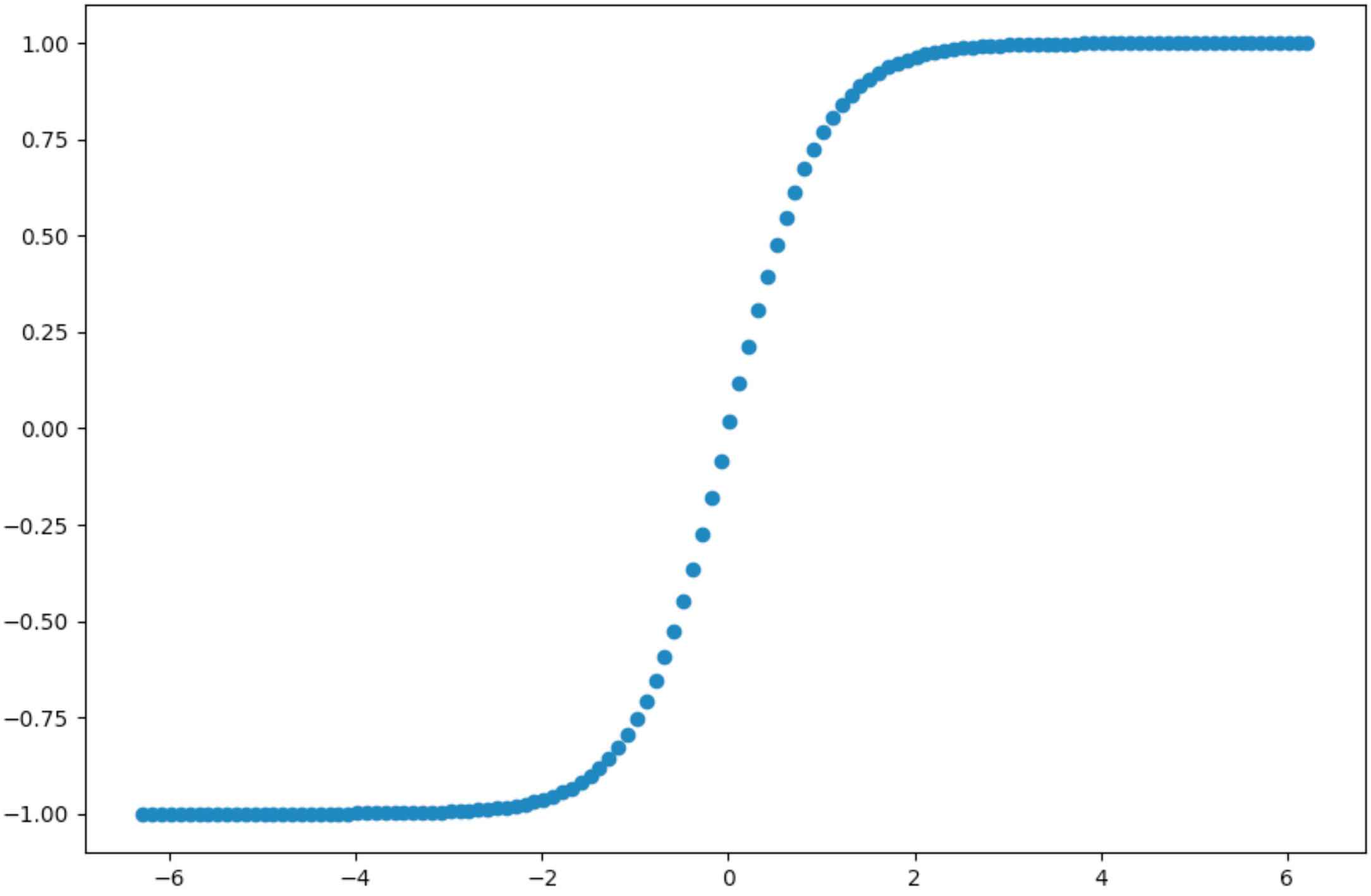
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$





y_hat = r(1:3)*tanh(a*tanh(a0*x(1,:)'+b0)+b)+r(4)



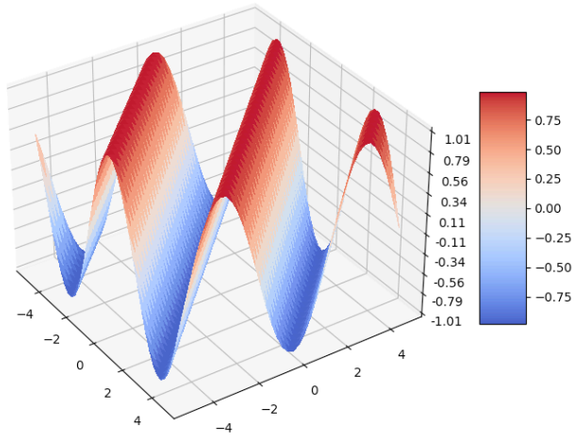

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−6
−4
−2
0
2
4
6

X1

X2

sin(x1+x2)

y



5 × 2

Len = 5

X1

X2

a0

Len = 2

5 × 1

5 × 1

3 × 5

5 × 1

tanh

a

3 × 1

3 × 1

1 × 3

1 × 1

1

b0

5 × 1

5 × 1

1

b

3 × 1

3 × 1

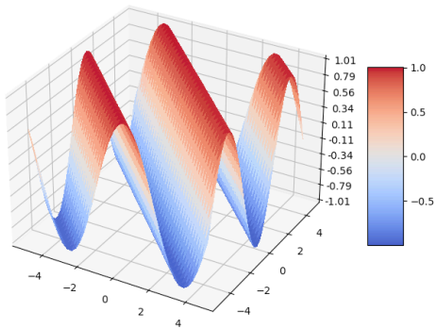
tanh

r(0:2)

r(3)

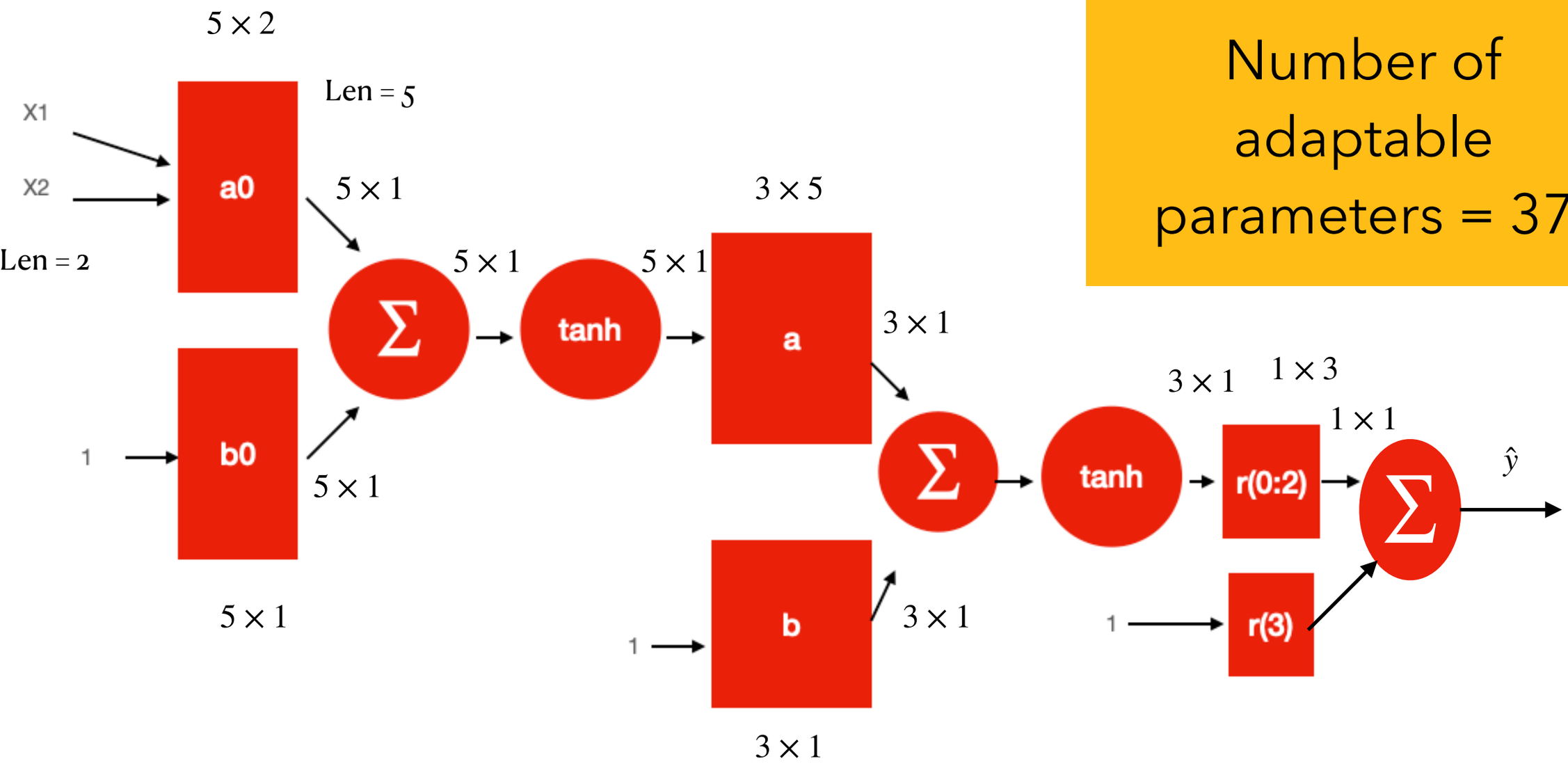
1

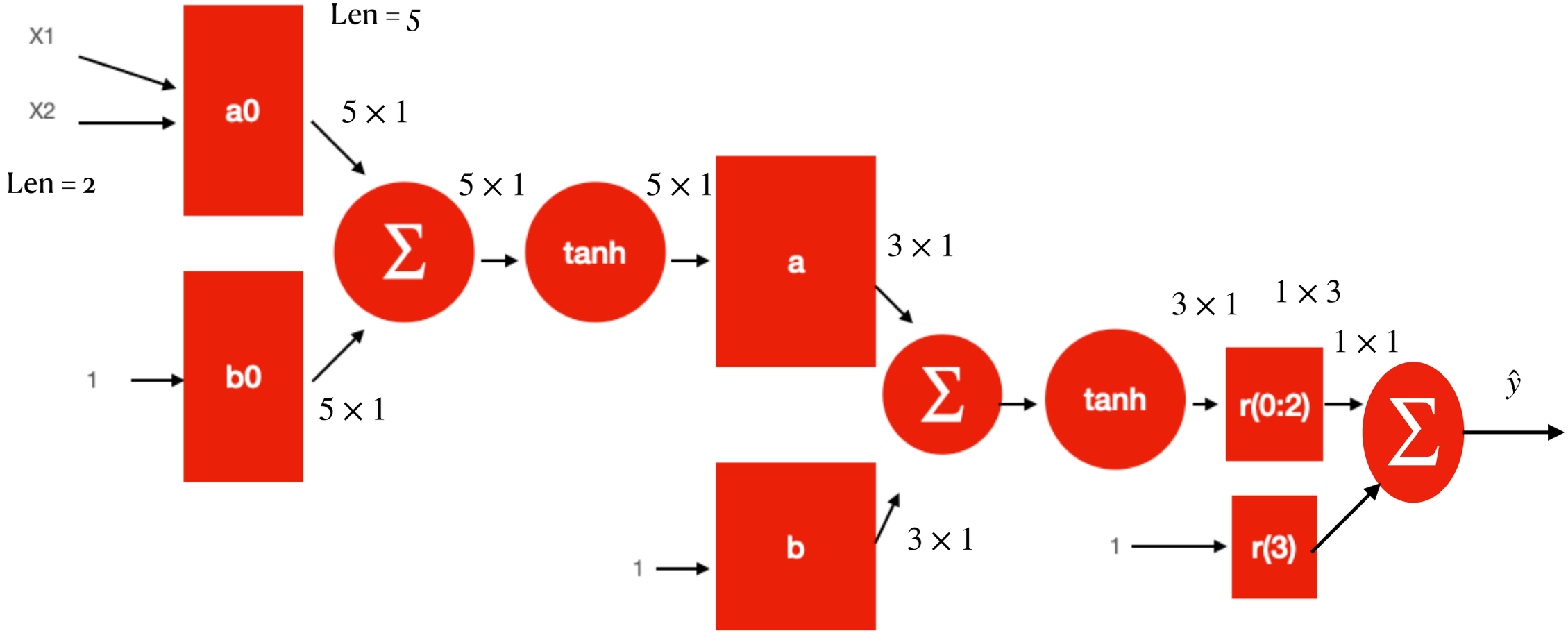
$\hat{y}$

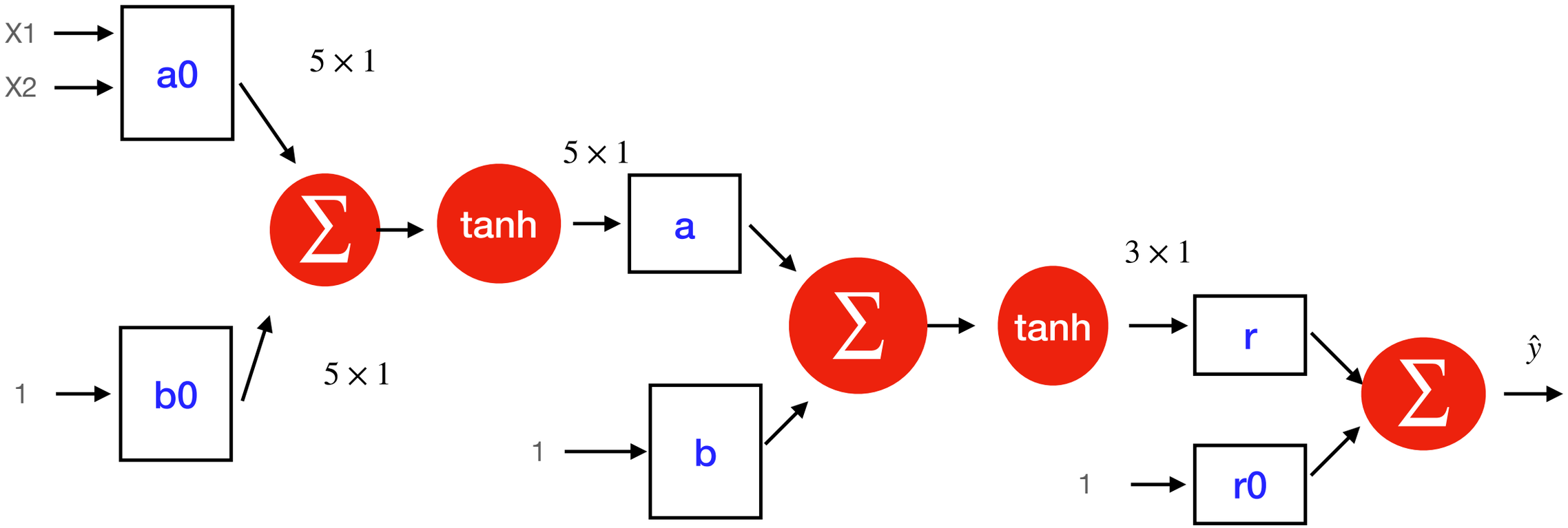


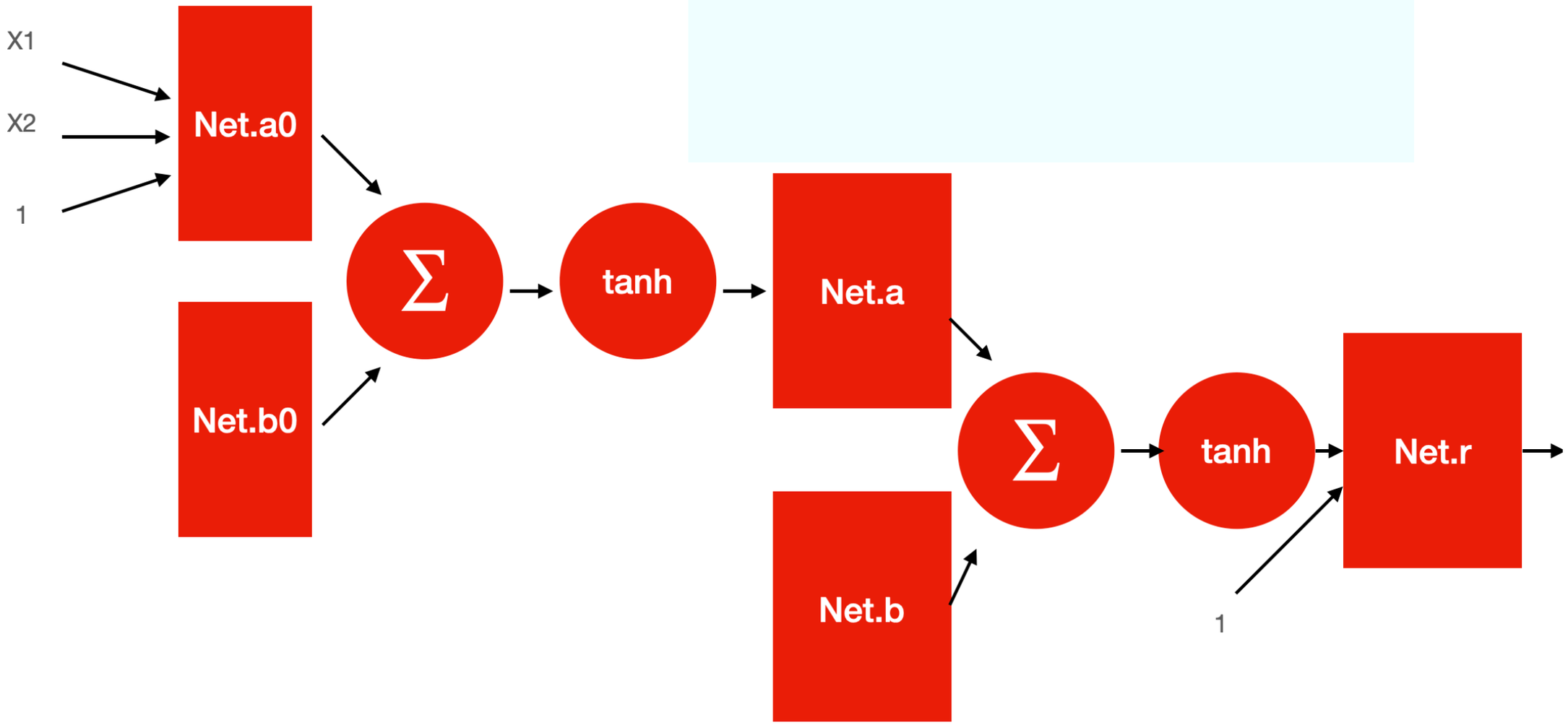
```
a0 = np.array([[-0.3918, -0.3918], [-0.0975, -0.0975],
               [-0.3304, -0.3303], [-0.2186, -0.2185],
               [-0.3472, -0.3472]])
b0 = np.array(
[[-3.5903], [0.1043], [1.9609], [1.1944], [-1.0927]])
a = np.array([[-0.6988, -0.1312, 0.5554, -2.1678, -0.1380],
              [-0.2736, -4.3798, 0.7643, 5.0048, -2.2072],
              [0.4992, -3.2731, -0.8272, 2.2502, 0.5677]])
b = np.array([[0.6347], [-0.3891], [-0.1289]])
r = np.array([-16.7872, 7.2905, 36.4077, 7.4048])
```

Number of adaptable parameters = 37

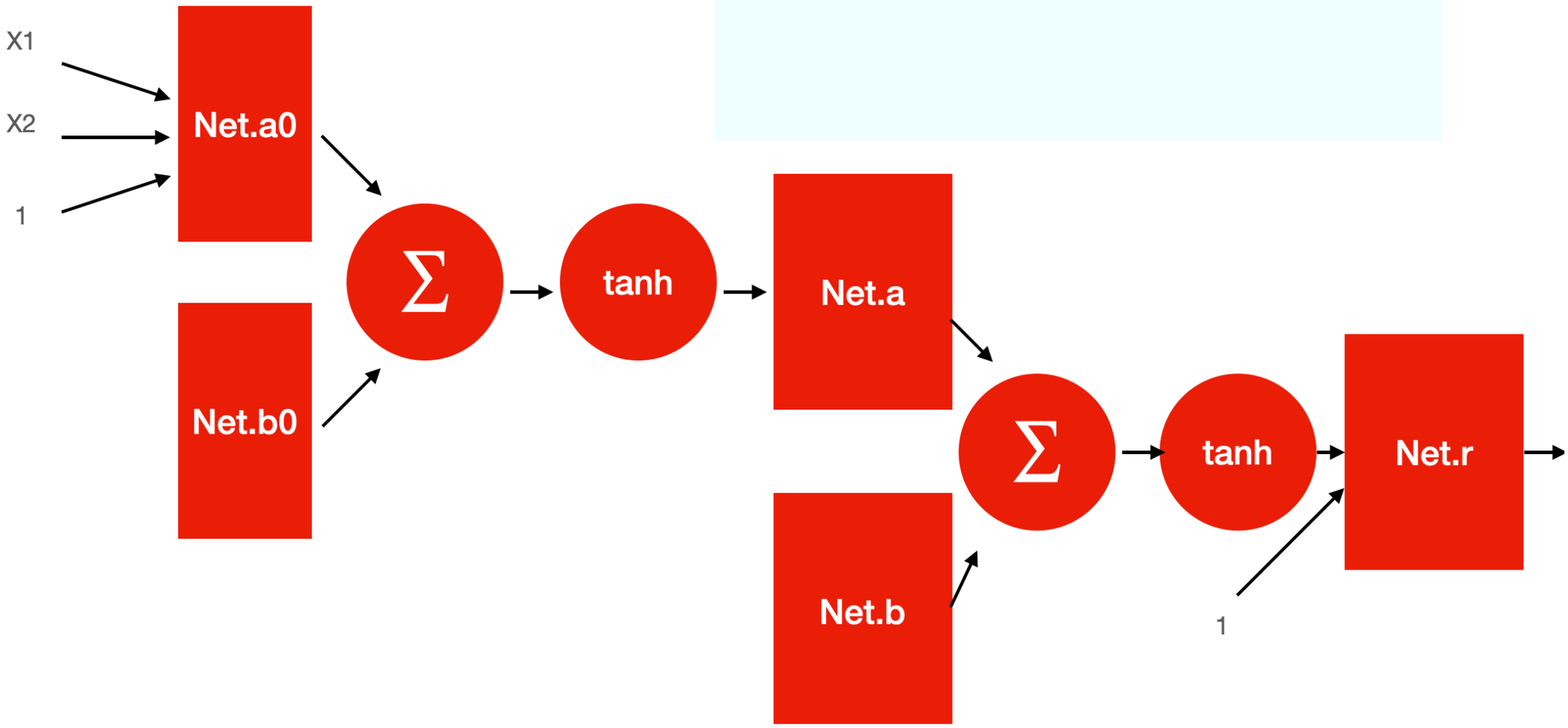



X1
X2
Len = 2
a0
Len = 5
5 × 1
5 × 1
5 × 1
tanh
a
3 × 1
1
b0
5 × 1
3 × 1
1 × 3
3 × 1
1 × 1
tanh
r(0:2)
$\hat{y}$
1
b
3 × 1
1
r(3)


X1
X2
a0
5 × 1
∑
tanh
5 × 1
a
∑
tanh
3 × 1
r
∑
ŷ
1
b0
5 × 1
1
b
1
r0




```
a0 = np.array([[-0.3918, -0.3918], [-0.0975, -0.0975],
               [-0.3304, -0.3303], [-0.2186, -0.2185],
               [-0.3472, -0.3472]])
b0 = np.array(
[[-3.5903], [0.1043], [1.9609], [1.1944], [-1.0927]])
a = np.array([[-0.6988, -0.1312, 0.5554, -2.1678, -0.1380],
              [-0.2736, -4.3798, 0.7643, 5.0048, -2.2072],
              [0.4992, -3.2731, -0.8272, 2.2502, 0.5677]])
b = np.array([[0.6347], [-0.3891], [-0.1289]])
r = np.array([-16.7872, 7.2905, 36.4077, 7.4048])
```




```
18      z = a0 @ x[0, :]
19      h0 = np.reshape(z, np.shape(b0)) + b0
20      v = np.tanh(h0)
21      h1 = a @ v + b
22      v2 = np.tanh(h1)
23      y_hat = r[0:3] @ v2 + r[3]
```

```
def mysin(x):
    a0 = np.array([[-0.3918, -0.3918], [-0.0975, -0.0975],
                   [-0.3304, -0.3303], [-0.2186, -0.2185],
                   [-0.3472, -0.3472]])
    b0 = np.array(
    [[-3.5903], [0.1043], [1.9609], [1.1944], [-1.0927]])
    a = np.array([[-0.6988, -0.1312, 0.5554, -2.1678, -0.1380],
                  [-0.2736, -4.3798, 0.7643, 5.0048, -2.2072],
                  [0.4992, -3.2731, -0.8272, 2.2502, 0.5677]])
    b = np.array([[0.6347], [-0.3891], [-0.1289]])
    r = np.array([-16.7872, 7.2905, 36.4077, 7.4048])
    z = a0 @ x[0, :]
    h0 = np.reshape(z, np.shape(b0)) + b0
    v = np.tanh(h0)
    h1 = a @ v + b
    v2 = np.tanh(h1)
    y_hat = r[0:3] @ v2 + r[3]
    return y_hat
```

```
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
shapeX = np.shape(X)
Z = np.zeros((shapeX[0],shapeX[1]))
v = np.random.rand(1,2)
for i in range(shapeX[0]):
    for j in range(shapeX[1]):
        v[0,0] = X[i,j]
        v[0,1]= Y[i,j]
        Z[i,j] = mysin(v)
```

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
from matplotlib.ticker import LinearLocator
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
# Make data.
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
Z = np.sin(X+Y)
# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
# Customize the z axis.
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
# A StrMethodFormatter is used automatically
ax.zaxis.set_major_formatter('{x:.02f}')
# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```