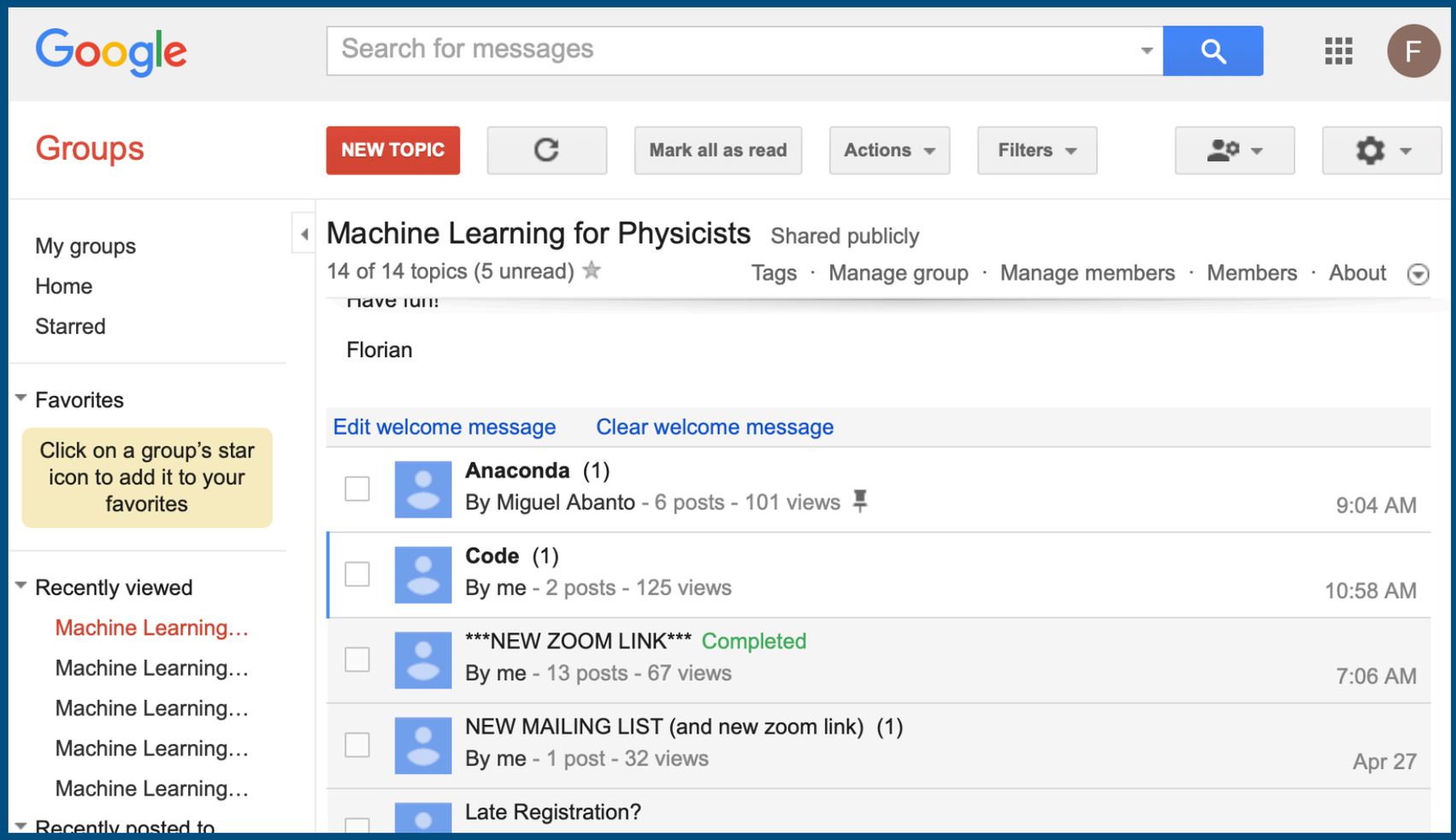
# Machine Learning for Physicists

Online Tutorials

Florian Marquardt
University of Erlangen-Nuremberg
& Max Planck Institute for the Science of Light
Florian.Marquardt@fau.de
http://machine-learning-for-physicists.org

(Image generated by a net with 20 hidden layers)

# Please use the forum for discussions & useful code & nice examples

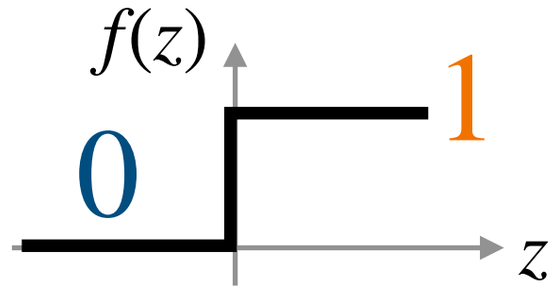



# Machine Learning for Physicists

# Quiz: single-layer neural networks

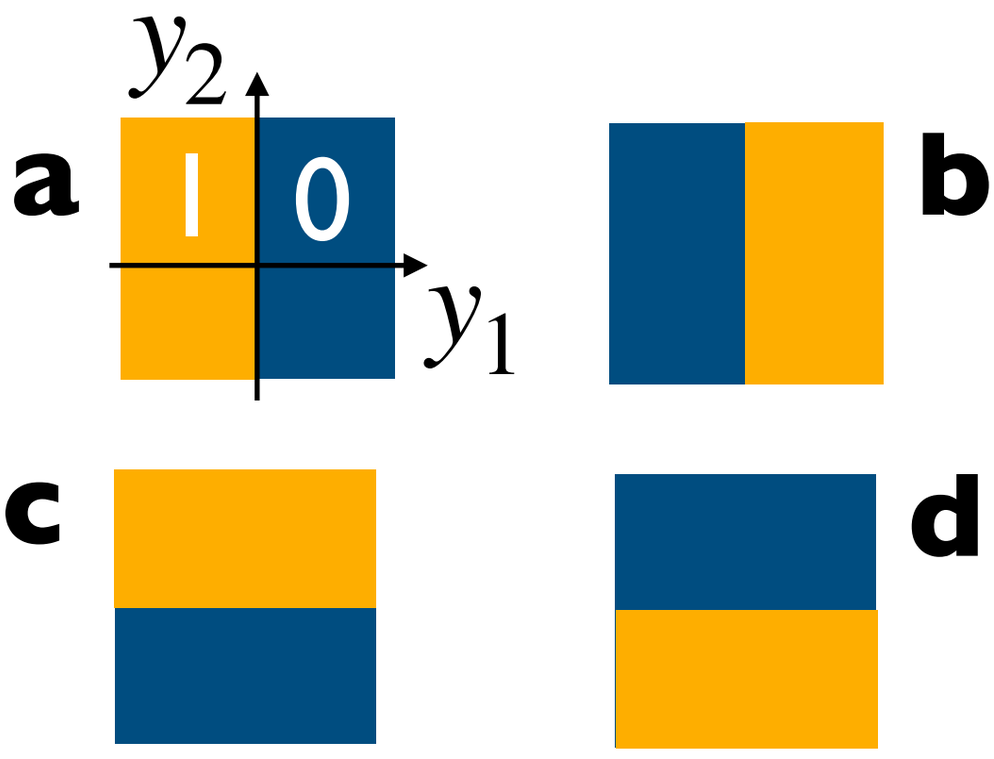
Machine Learning for Physicists

$f(z)$ is the step function:



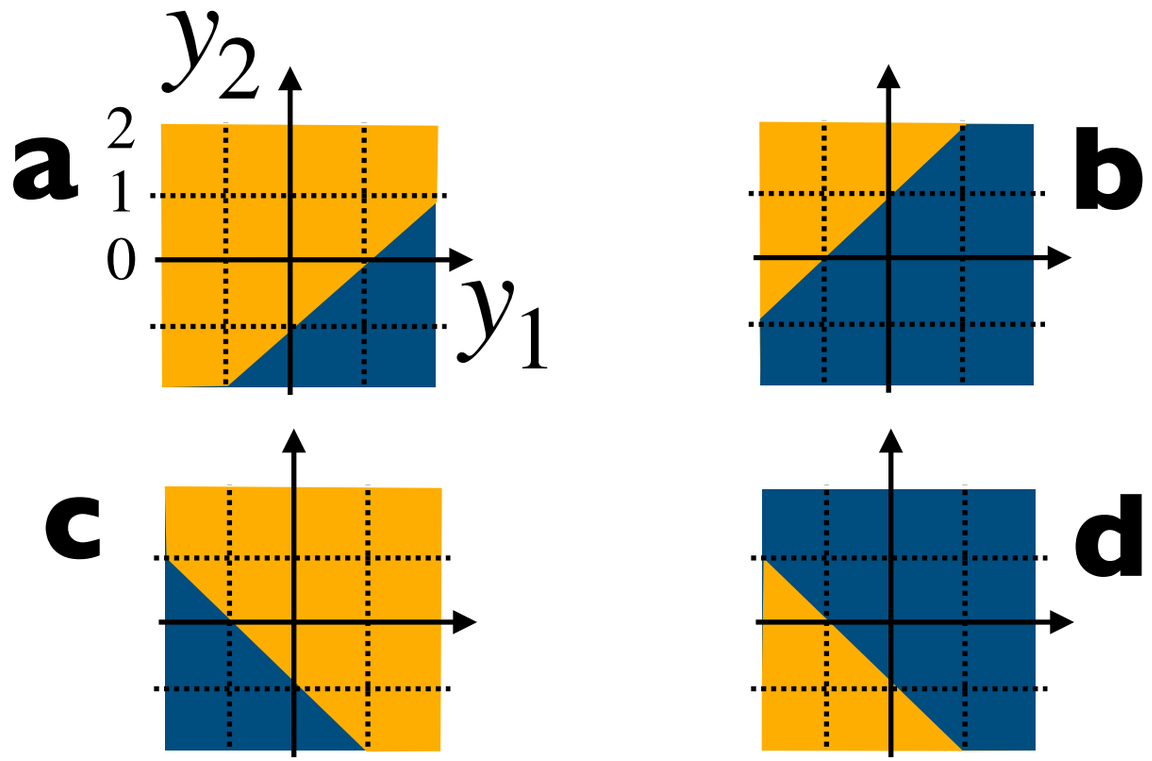

1

Which of these outputs is generated by $y^{\text{out}} = f(-y_2)$ ?

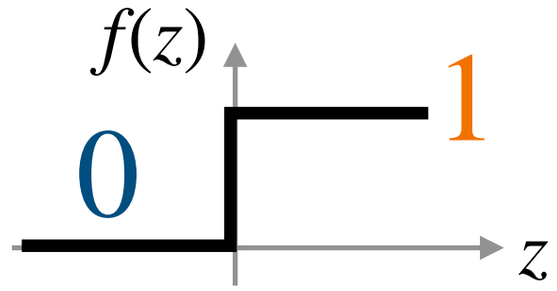



2

…and for $y^{\text{out}} = f(1 - y_1 + y_2)$ ?

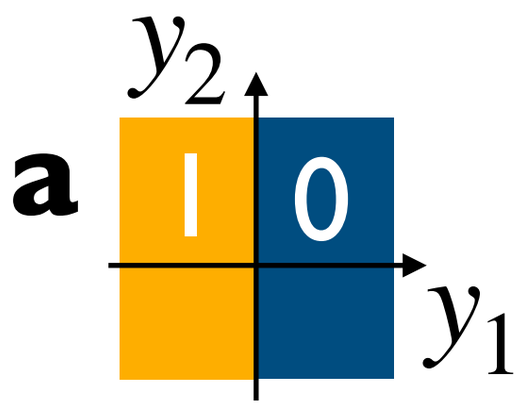


$f(z)$ is the step function:

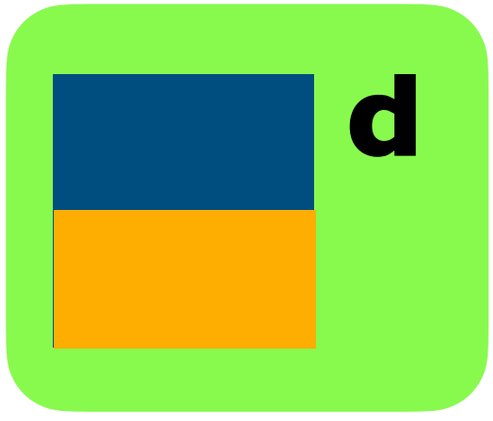



1

Which of these outputs is generated by $y^{\text{out}} = f(-y_2)$ ?

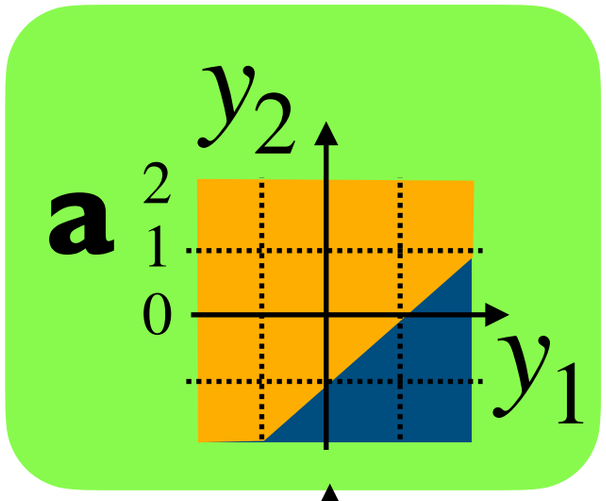


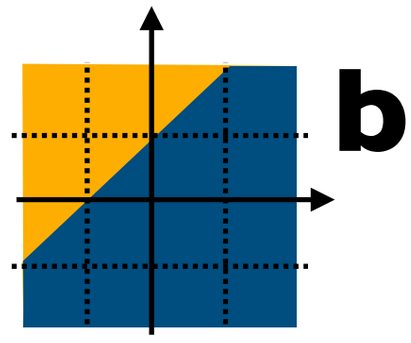


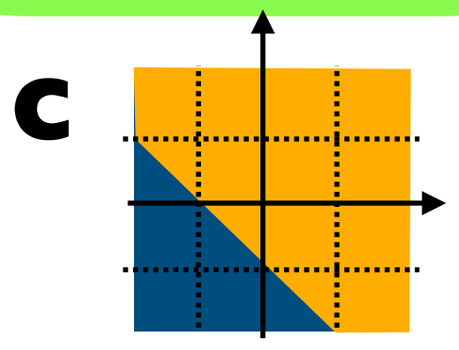



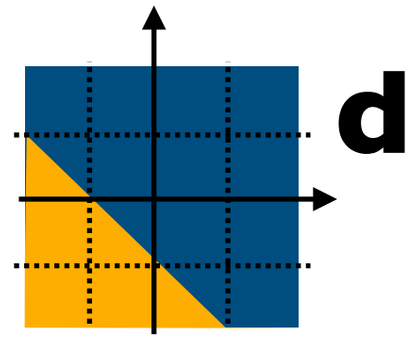

2

…and for $y^{\text{out}} = f(1 - y_1 + y_2)$ ?

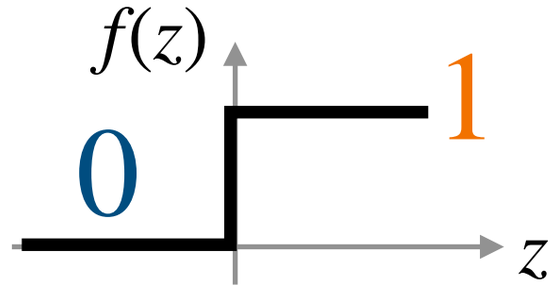


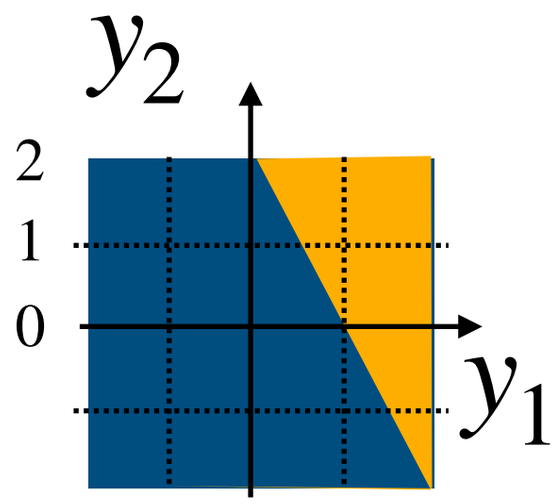



$f(z)$ is the step function:




**1** How can we obtain this output?



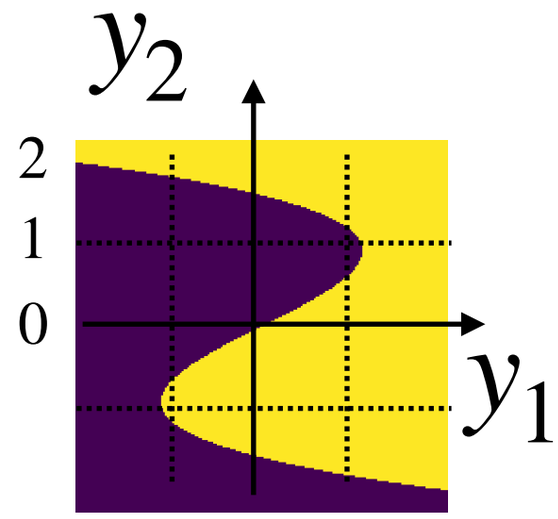

**a** $y^{\text{out}} = f(y_1 + y_2 - 2)$

**b** $y^{\text{out}} = f(2y_1 + y_2 - 2)$

**c** $y^{\text{out}} = f(2 - 2y_1 + y_2)$

**d** $y^{\text{out}} = f(y_1 - 2y_2)$

**2** …and this one?



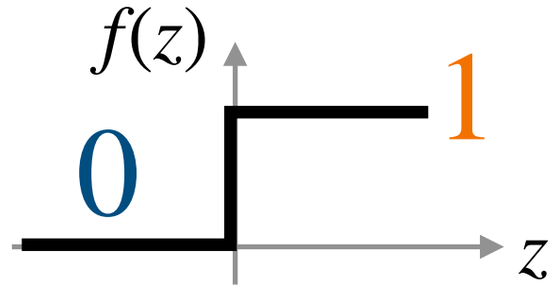

**a** $y^{\text{out}} = f(-2y_2 + y_1)$

**b** $y^{\text{out}} = f(y_2^3 - 2y_2 - y_1)$

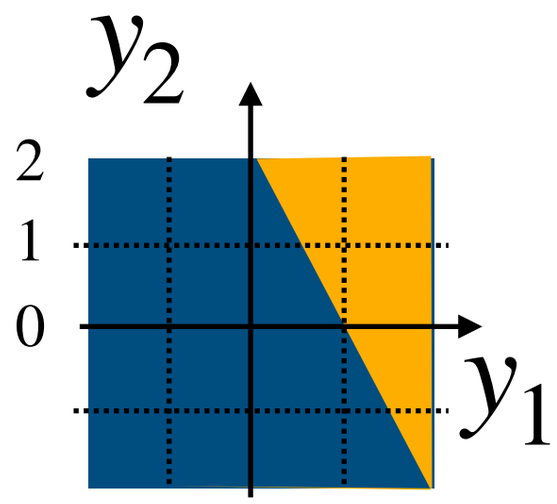
**c** $y^{\text{out}} = f(y_2^3 - 2y_2 + y_1)$

**d** $y^{\text{out}} = f(-y_2^3 - 2y_2 + y_1)$

$f(z)$ is the step function:




**1** How can we obtain this output?



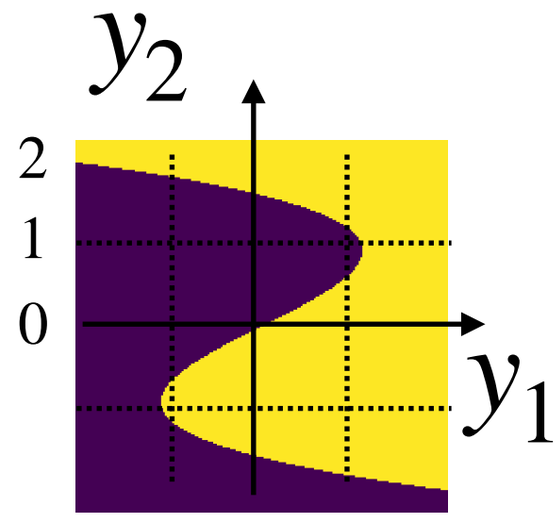

**a** $y^{\text{out}} = f(y_1 + y_2 - 2)$

**b** $y^{\text{out}} = f(2y_1 + y_2 - 2)$

**c** $y^{\text{out}} = f(2 - 2y_1 + y_2)$

**d** $y^{\text{out}} = f(y_1 - 2y_2)$

**2** …and this one?




**a** $y^{\text{out}} = f(-2y_2 + y_1)$

**b** $y^{\text{out}} = f(y_2^3 - 2y_2 - y_1)$

**c** $y^{\text{out}} = f(y_2^3 - 2y_2 + y_1)$

**d** $y^{\text{out}} = f(-y_2^3 - 2y_2 + y_1)$

# Quiz: python

Machine Learning for Physicists

1

```
for j in range(2):
    print(j)
```

**a** 0 1

**b** 1 2

**c** 0 1 2

2

```
def f(n):
    if n<=2:
        return(n+f(n+1))
    else:
        return(42)
```

f(1)=?

**a** 85

**b** 45

**c** 44

**d** 42

**e** 3

1

```
for j in range(2):
    print(j)
```

**a** 0 1

**b** 1 2

**c** 0 1 2

2

```
def f(n):
    if n<=2:
        return(n+f(n+1))
    else:
        return(42)

f(1)=?
```

**a** 85

**b** 45

**c** 44

**d** 42

**e** 3

f(1)=1+f(2)=1+2+f(3)=1+2+42

```
a=np.array([[11,22],[33,44]])
```

1 a[0,1]=?

**a** 11

**b** 22

**c** 33

**d** 44

2 a[1]=?

**a** 22

**b** [11,22]

**c** [33,44]

**d** [22,44]

3 a[:,1]=?

**a** [22,44]

**b** [11,33]

**c** [33,44]

**d** [11,22]

```
a=np.array([[11,22],[33,44]])
```

1 a[0,1]=?

**a** 11 **c** 33

**b** 22 **d** 44

2 a[1]=?

**a** 22

**b** [11,22]

**c** [33,44]

**d** [22,44]

3 a[:,1]=?

**a** [22,44]

**b** [11,33]

**c** [33,44]

**d** [11,22]

$$\begin{matrix} a_{0,0} & a_{0,1} \\ \begin{pmatrix} 11 & 22 \\ 33 & 44 \end{pmatrix} \\ a_{1,0} & a_{1,1} \end{matrix}$$

```
a=np.array([[11,22],[33,44]])
```

**1** `a.flatten()=?`

**a** `[11,33,22,44]`

**b** `[11,22,33,44]`

**c** `[[11],[33],[22],[44]]`

**2**

```
j0=np.array([1,0,1])
j1=np.array([0,0,1])
a[j0,j1]=?
```

**a** `[11,33,44]`

**b** error

**c** `[22,11,44]`

**d** `[33,11,44]`

**3**

```
b=np.array([19,50])
a[:,0]=b[:]
a=?
```

**a** `[[11,22],[19,50]]`

**b** `[[19,50],[33,44]]`

**c** `[[19,22],[50,44]]`

**d** `[[11,19],[33,50]]`

```
a=np.array([[11,22],[33,44]])
```

**1** `a.flatten()=?`

**a** `[11,33,22,44]`

**b** `[11,22,33,44]`

**c** `[[11],[33],[22],[44]]`

**2**

```
j0=np.array([1,0,1])
j1=np.array([0,0,1])
a[j0,j1]=?
```

**a** `[11,33,44]`

**b** error

**c** `[22,11,44]`

**d** `[33,11,44]`

**3**

```
b=np.array([19,50])
a[:,0]=b[:]
a=?
```

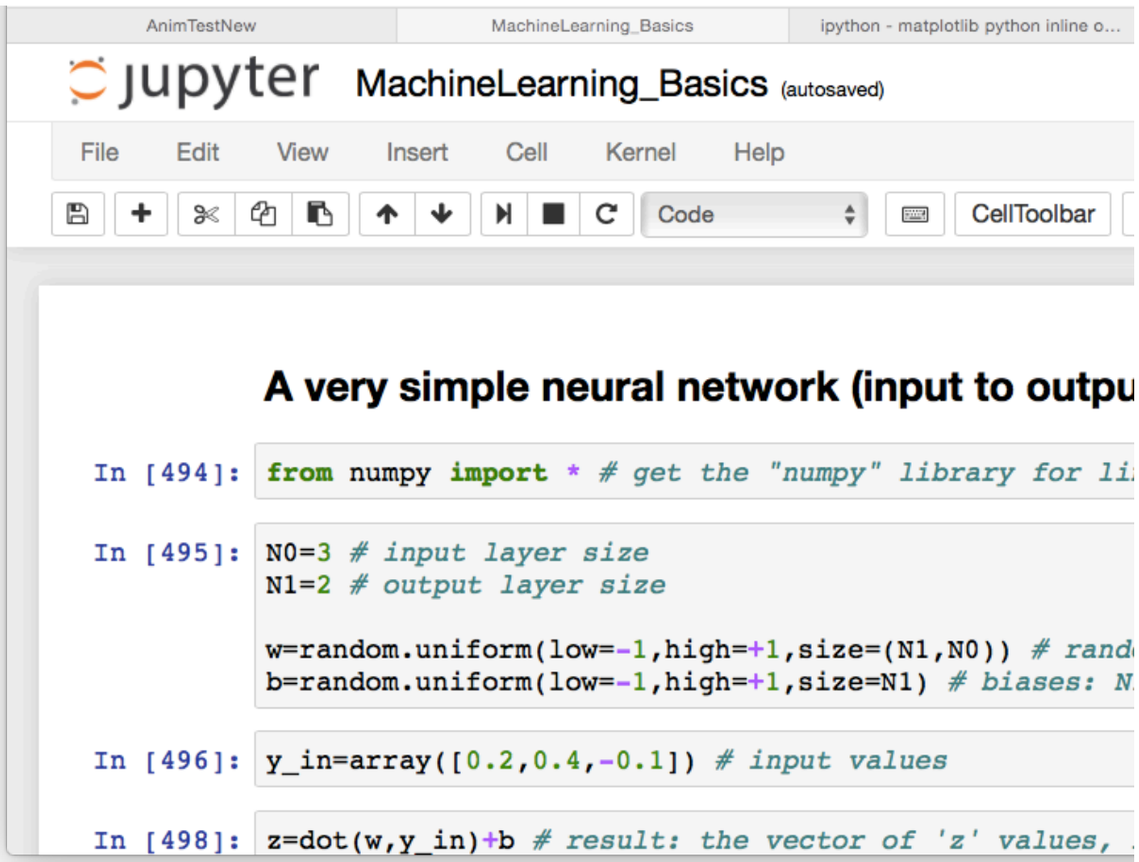
**a** `[[11,22],[19,50]]`

**b** `[[19,50],[33,44]]`

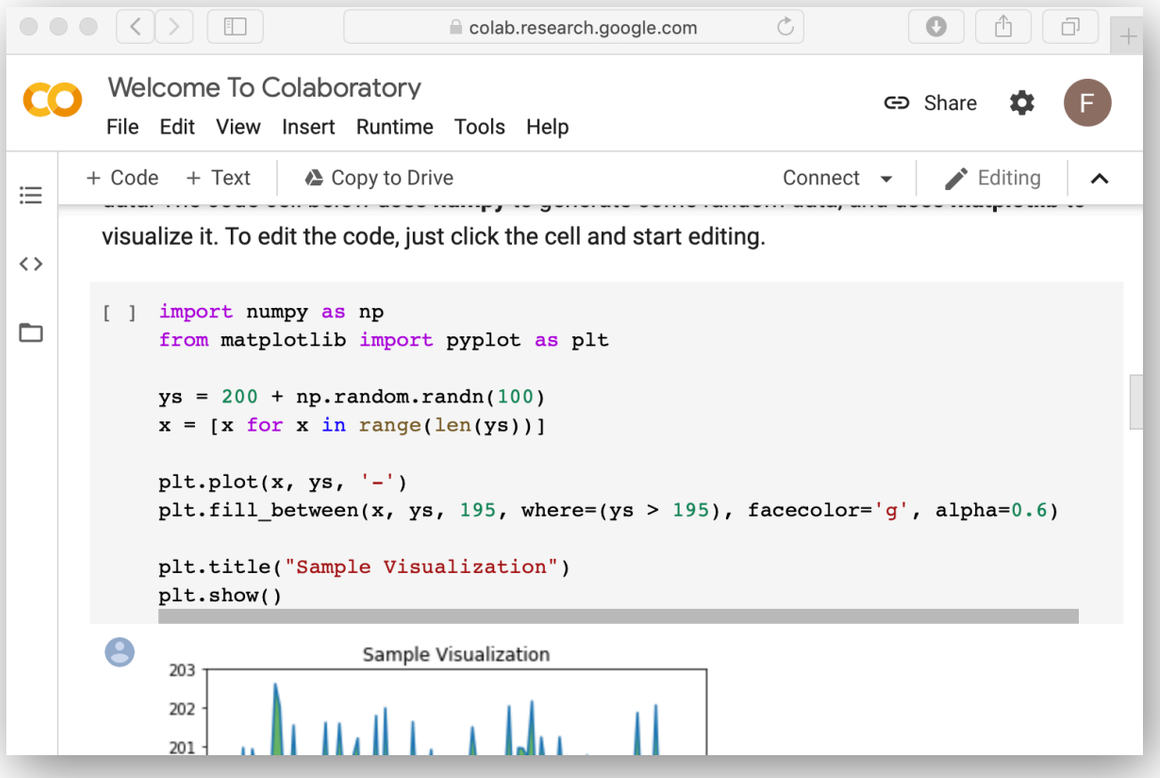
**c** `[[19,22],[50,44]]`

**d** `[[11,19],[33,50]]`

# Jupyter




# Colaboratory



# Practice session: Visualizing neural networks

Machine Learning for Physicists

**Code (jupyter notebook):** 01_MachineLearning_Basics_NeuralNetworksPython.ipynb
(or download code as pure python script)

see: website/course overview/lecture 1

This notebook shows how to calculate the forward-pass through a neural network in pure python, and how to illustrate the results for randomly initialized deep neural networks (as shown in the lecture).

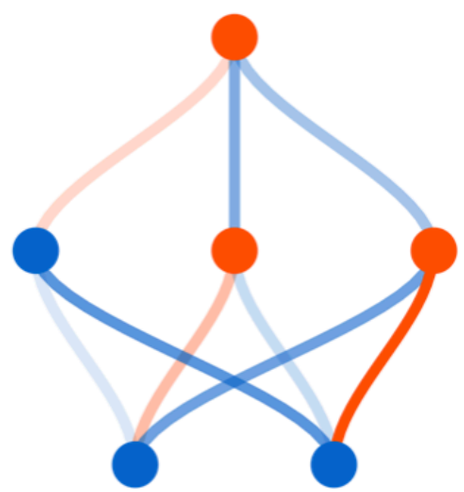
**Notebooks for tutorials:**
Tutorial: Network Visualization
Tutorial: Curve Fitting

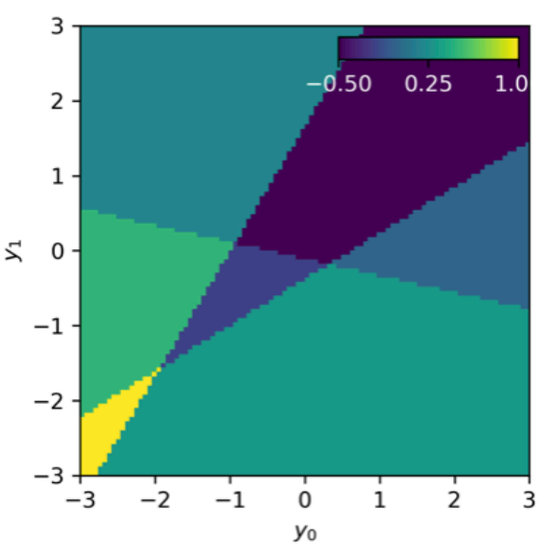
Settings Help

01_tutoria 01_Machi

Code Python 3

```
0.5], # biases of 3 hi
s for output neuron
],
ctivations=[ 'jump', # activation fo
             'linear' # activation for
           ],
y0range=[-3,3],y1range=[-3,3])
```





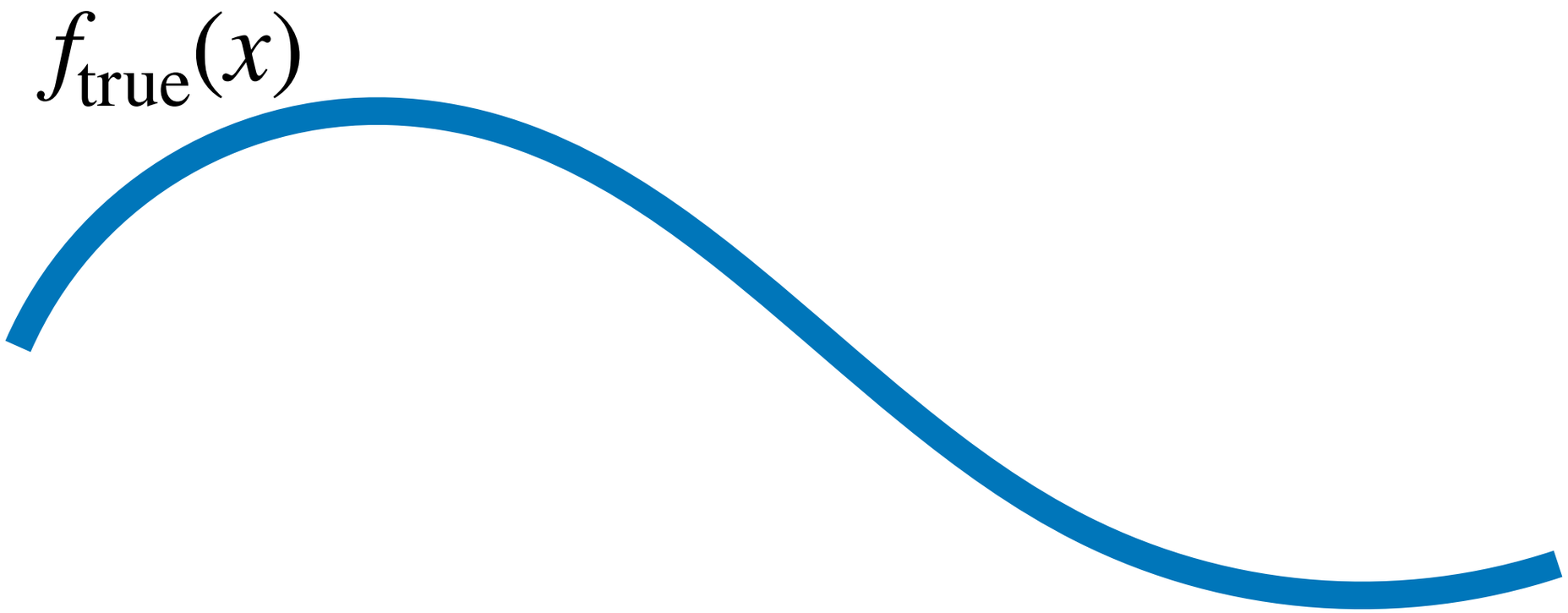

0 $_ 7 Python Mode: Ln 1, 01_tutorial_NetworkVisua

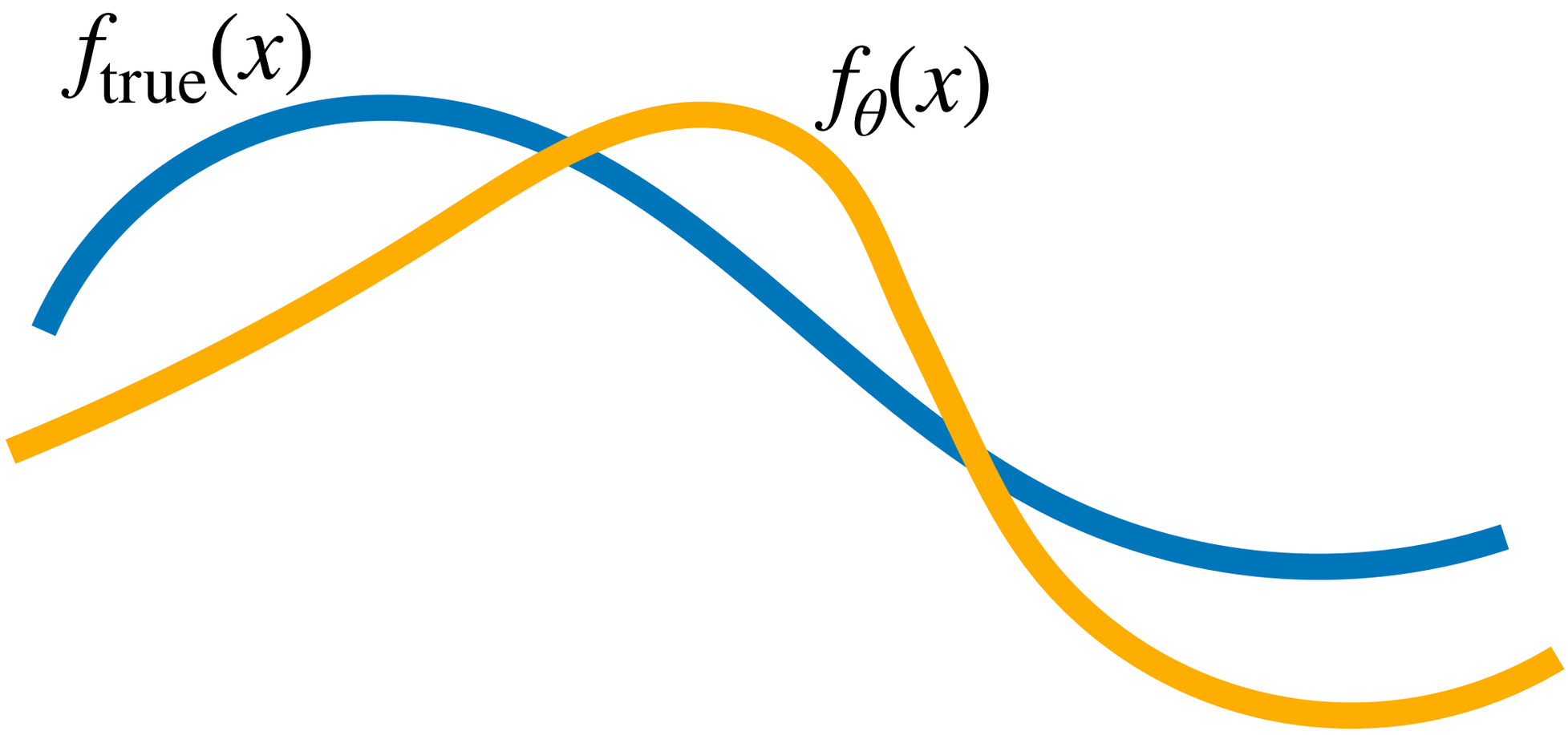
(1) Try to construct a square!

(2) What happens when you take a sigmoid hidden layer and a 'jump' (stepfunction) output layer? Which shapes can you construct?

# Practice session: Nonlinear curve fitting

Machine Learning for Physicists


$f_{\text{true}}(x)$




e.g. $f_\theta(x) = \dfrac{\theta_0}{(x - \theta_1)^2 + 1}$

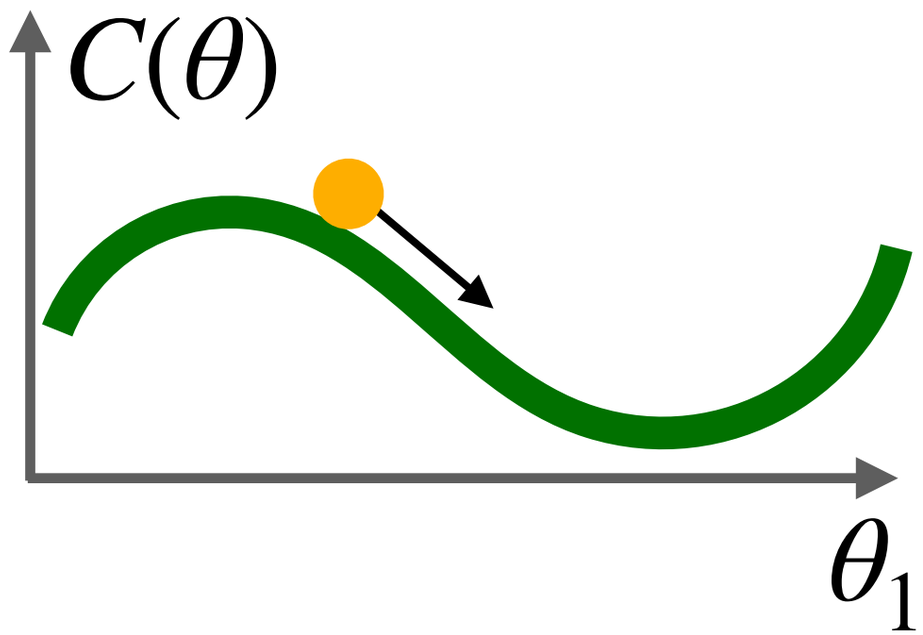
"least squares fitting":
consider quadratic deviation

$$C(\theta) = \frac{1}{2} \left\langle (f_\theta(x) - f_{\text{true}}(x))^2 \right\rangle$$

(average over x)

gradient descent ("down the hill"):

$$\delta\theta_j = -\eta \frac{\partial C}{\partial \theta_j}$$




stochastic: sample x

**Code (jupyter notebook):** 01_MachineLearning_Basics_NeuralNetworksPython.ipynb
(or download code as pure python script)

## see: website/course overview/lecture 1

This notebook shows how to calculate the forward-pass through a neural network in pure python, and how to illustrate the results for randomly initialized deep neural networks (as shown in the lecture).

**Notebooks for tutorials:**

Tutorial: Network Visualization

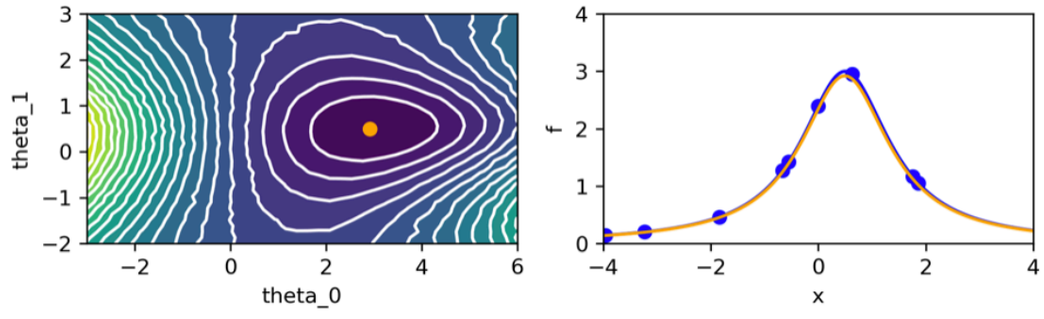
Tutorial: Curve Fitting

Settings Help

_Network 01_MachineLearnin

Markdown Python 3

```
ax[1].plot(x_sweep,f(theta,x_sweep),color="ora
ax[1].set_xlim(-4,4)
ax[1].set_ylim(0.0,4.0)
ax[1].set_xlabel("x")
ax[1].set_ylabel("f")

plt.show()
sleep(0.3)
```



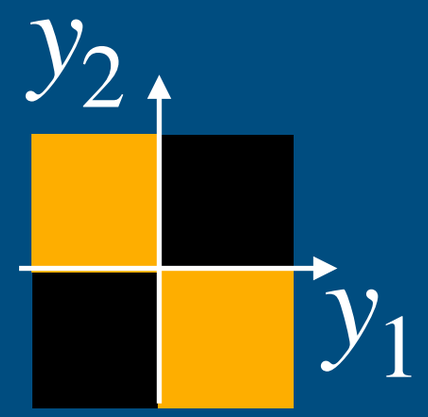

Your own examples!

0 $_ 7 Python 3 | Mode: Ln 1, 01_tutorial_CurveFitting.i

# Suggested Homework

## Machine Learning for Physicists

1. * Implement a network that computes XOR




2. Implement a network that approximately computes XOR, with 1 hidden layer(!)

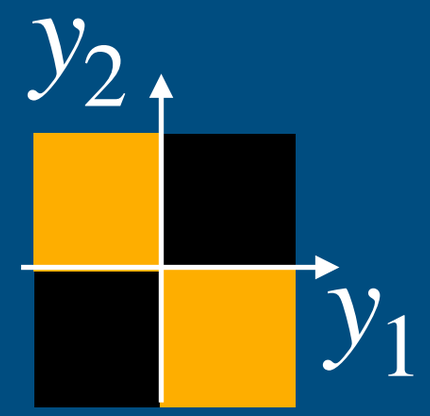
* minimum

3. * Visualize the results of intermediate layers in a multi-layer randomly initialized NN
4. What happens when you change the spread of the random weights?
5. Explore cases of curve fitting where there are several (non-equivalent) local minima. Is sampling noise helpful?

# Hints & Solutions for Lecture 1 Homework

Machine Learning for Physicists

1. Implement a network that computes XOR*
2. Implement a network that approximately computes XOR, with 1 hidden layer(!)




* minimum

# Machine Learning for Physicists

"warm up" AND function

+1 for (y0,y1)=(1,1) and 0 for other combinations like (1,0)

```
y_out=f(y0+y1-1.5)
```

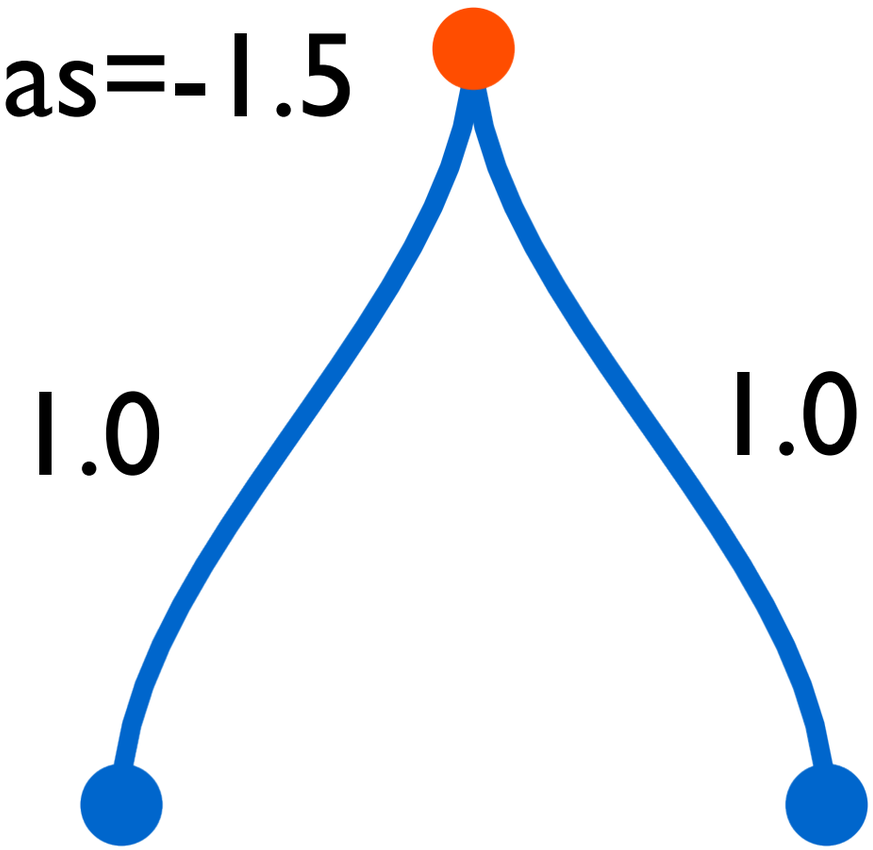
f = step function

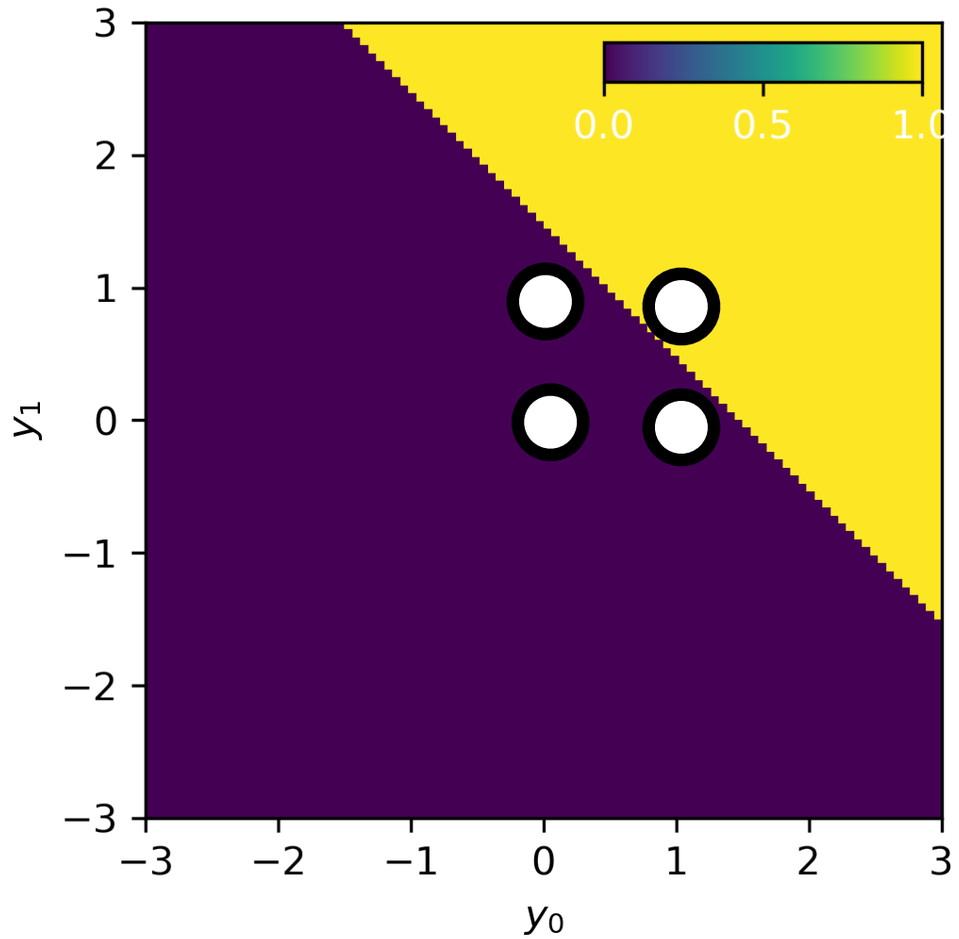
"warm up" AND function

+1 for (y0,y1)=(1,1) and 0 for other combinations like (1,0)

```
y_out=f(y0+y1-1.5)
```

bias=-1.5

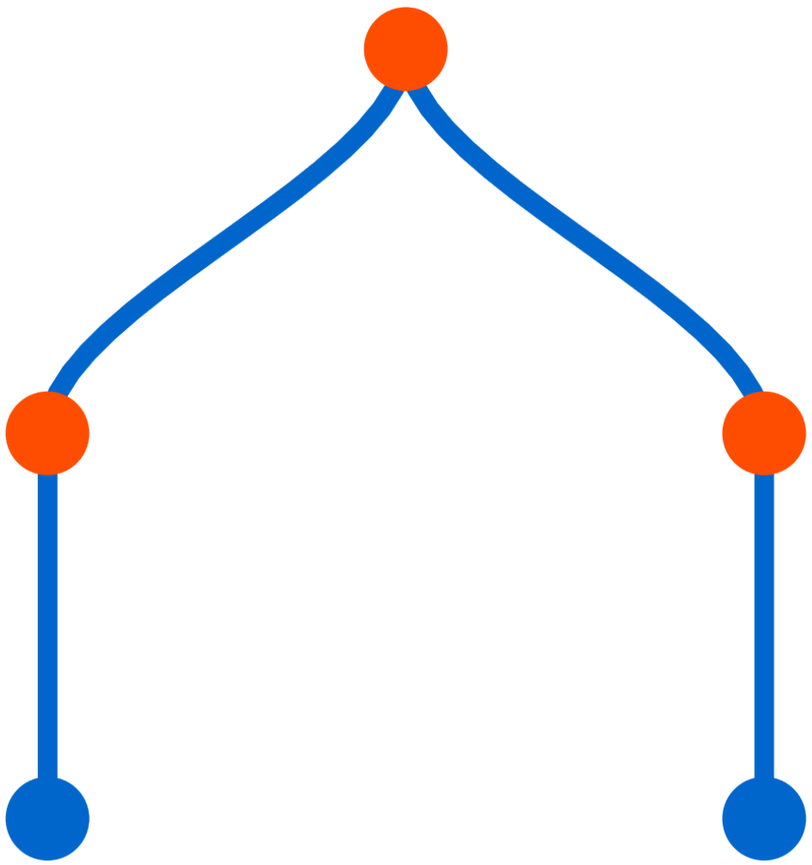


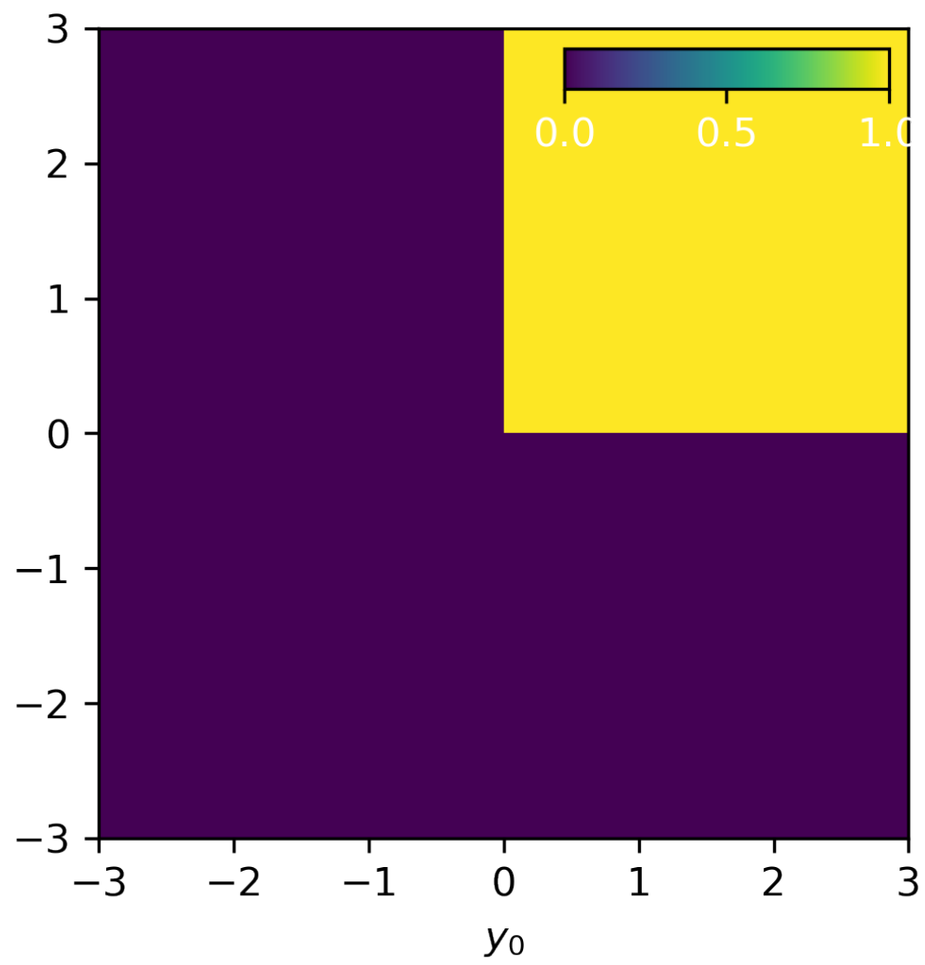




```
weights=[ [    [1.0,1.0]     ] ],

biases=[ [-1.5] ]
```

# "warm up" alternative AND function

## +1 exactly for y0>0 AND y1>0



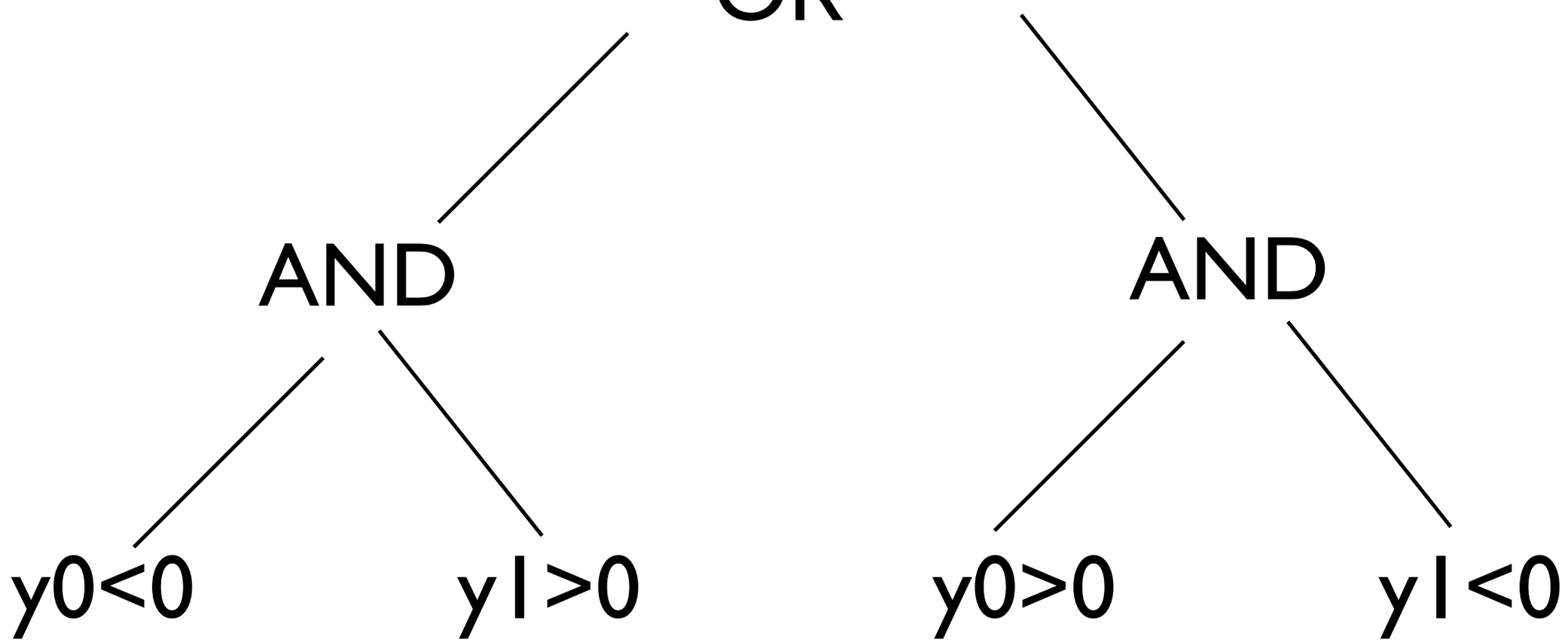



```
weights=[ [ [1.0,0.0], [0.0,1.0] ],
[ [1.0,1.0] ] ],
biases=[  [0.0,0.0], [-1.5] ]
```

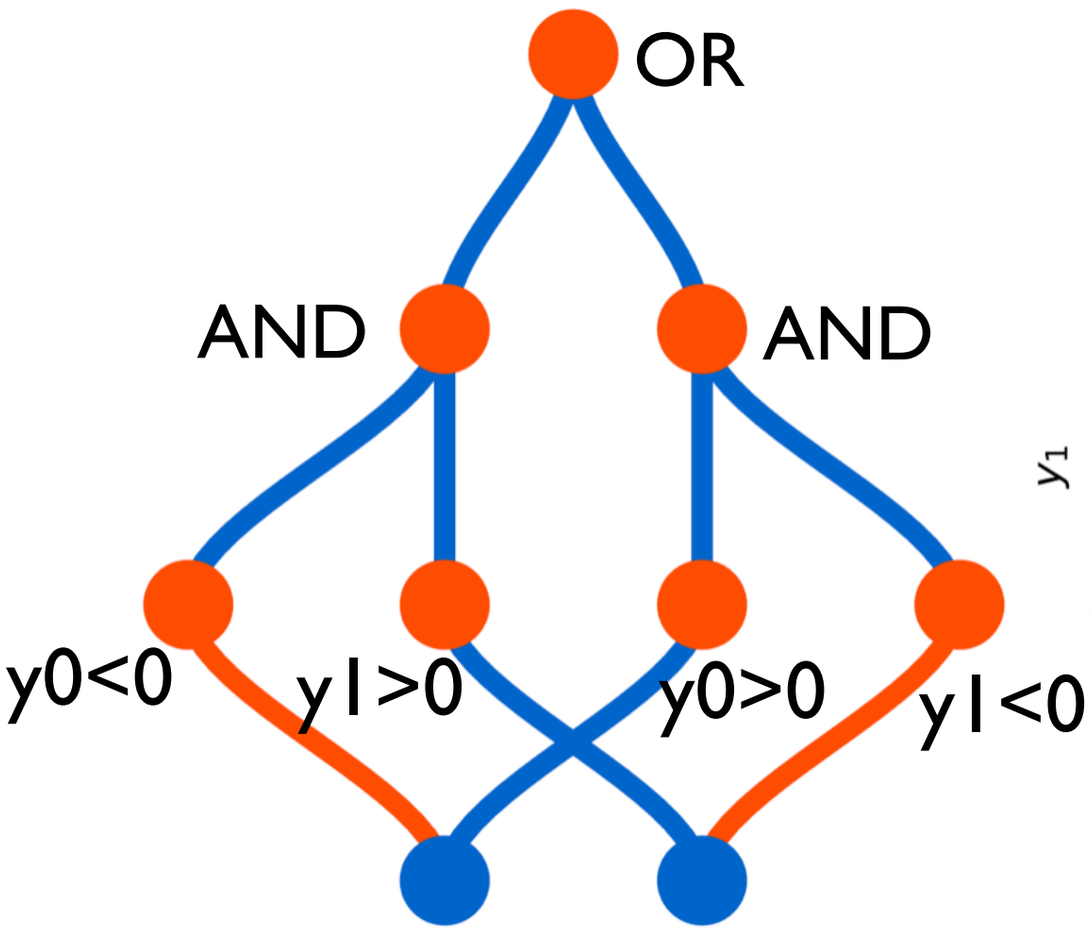
# the XOR function

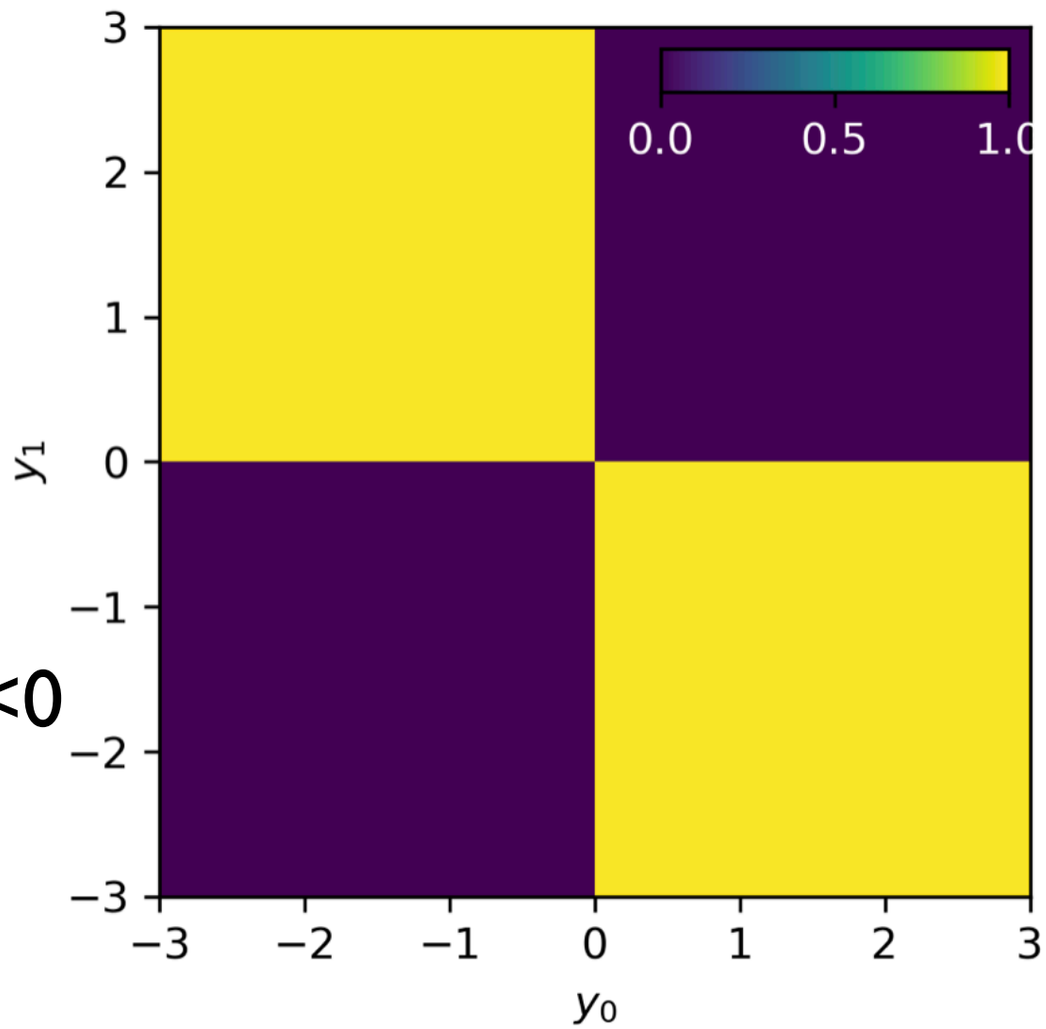
"exclusive or": +1 for y0*y1<0

(y0<0 AND y1>0) OR (y0>0 AND y1<0)

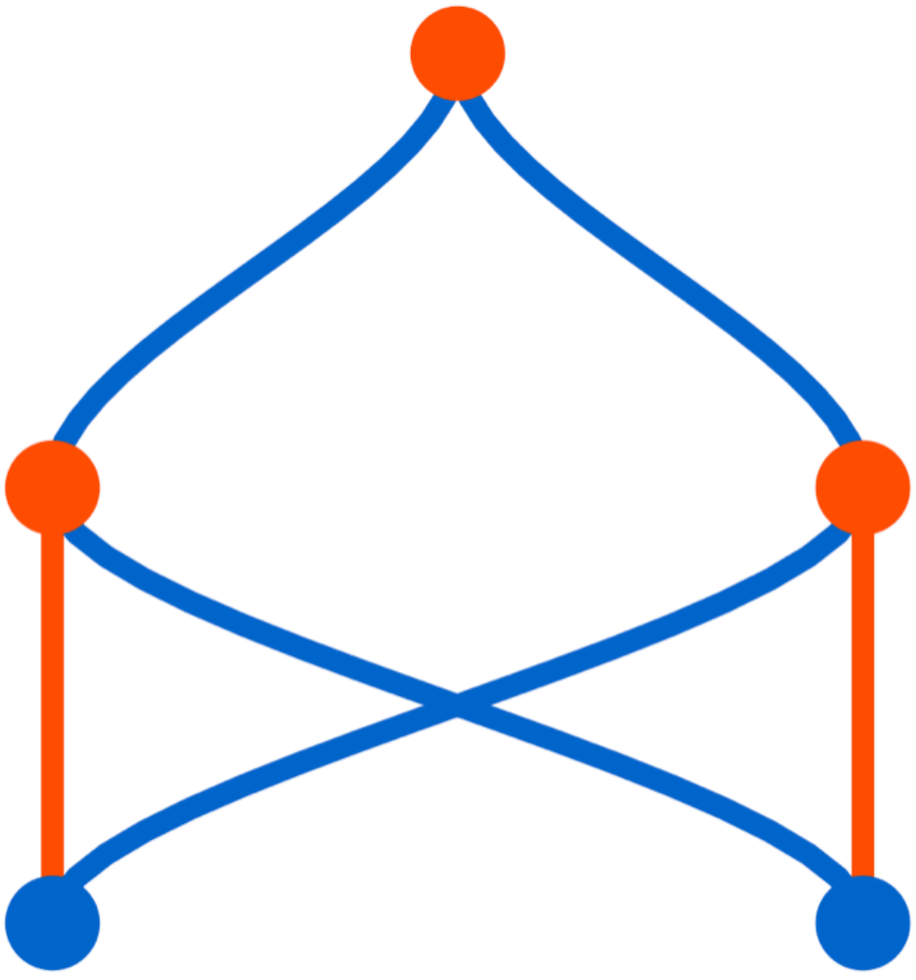


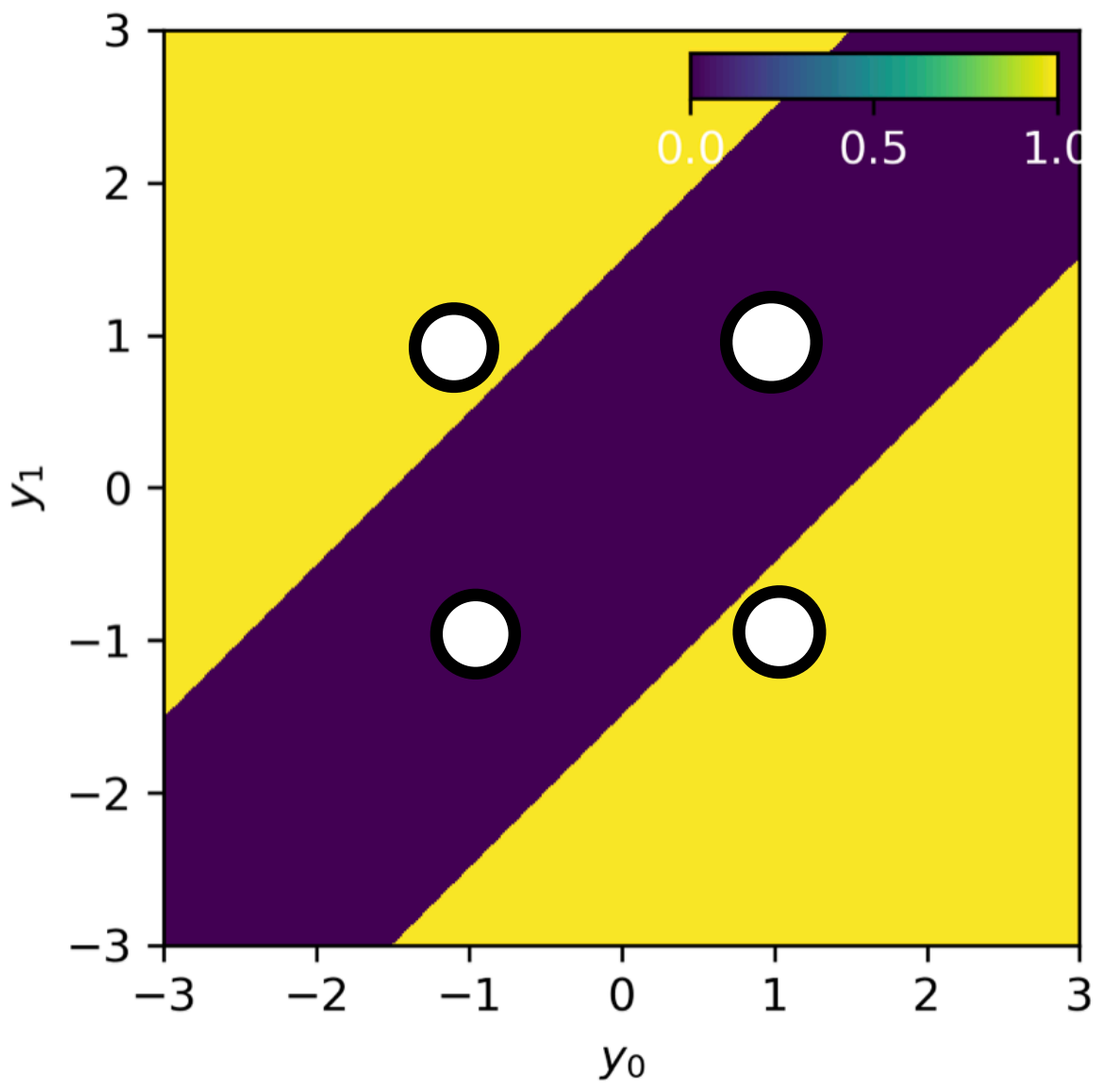
```
# XOR = (y0<0 AND y1>0) OR (y0>0 AND y1<0)
# 4 neurons in first hidden layer
# 2 neurons in second hidden layer

visualize_network(weights=[ [ [-1.0,0.0],[0,1],
                              [1,0],[0,-1] ],
                            [ [1,1,0,0],[0,0,1,1] ],
                           [  [1,1]    ]  ],
                  biases=[ [0,0,0,0], [-1.5,-1.5], [0] ],
                  activations=[ 'jump', 'jump', 'linear' ],
                  y0range=[-3,3],y1range=[-3,3], M=400)
```

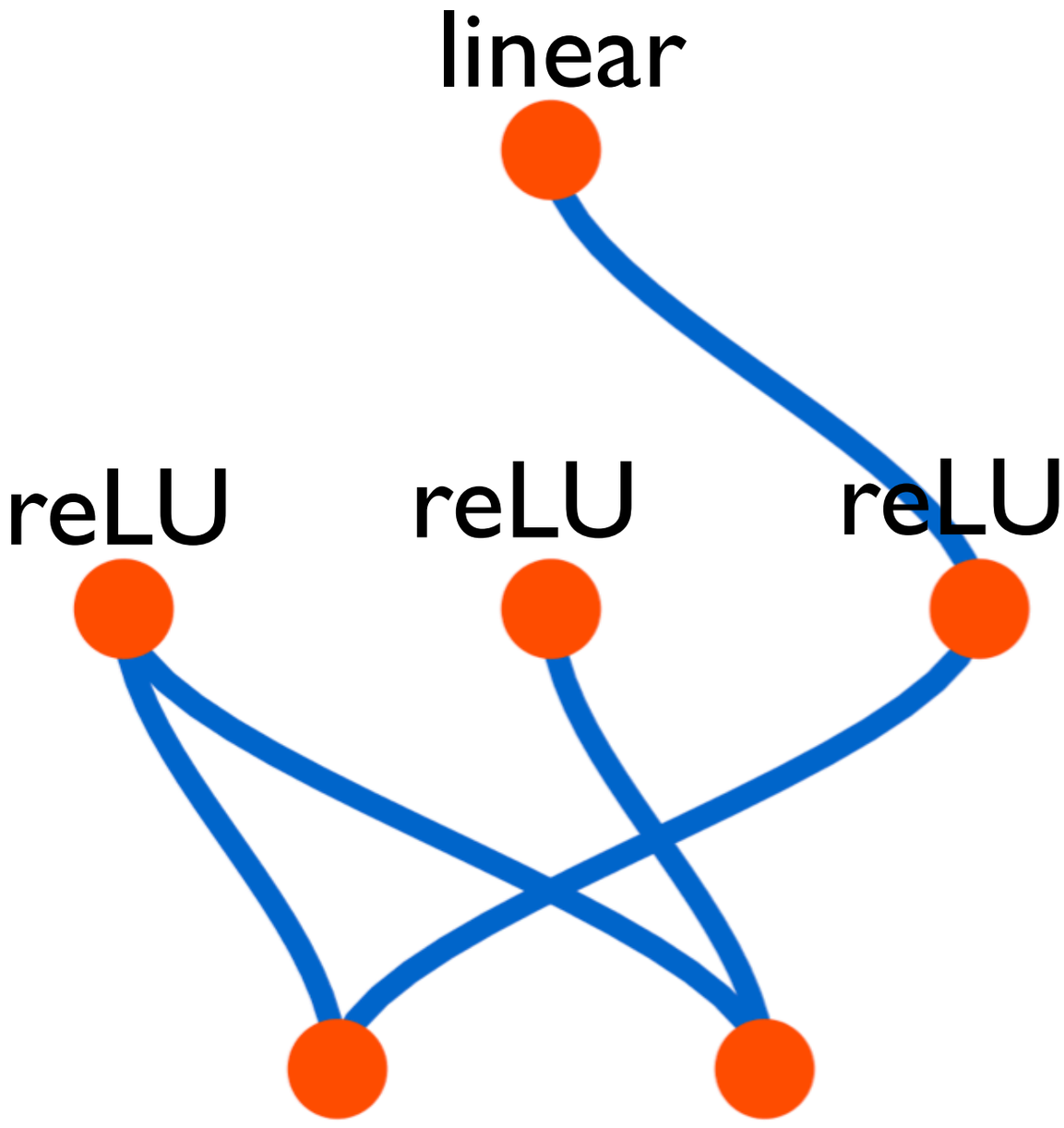


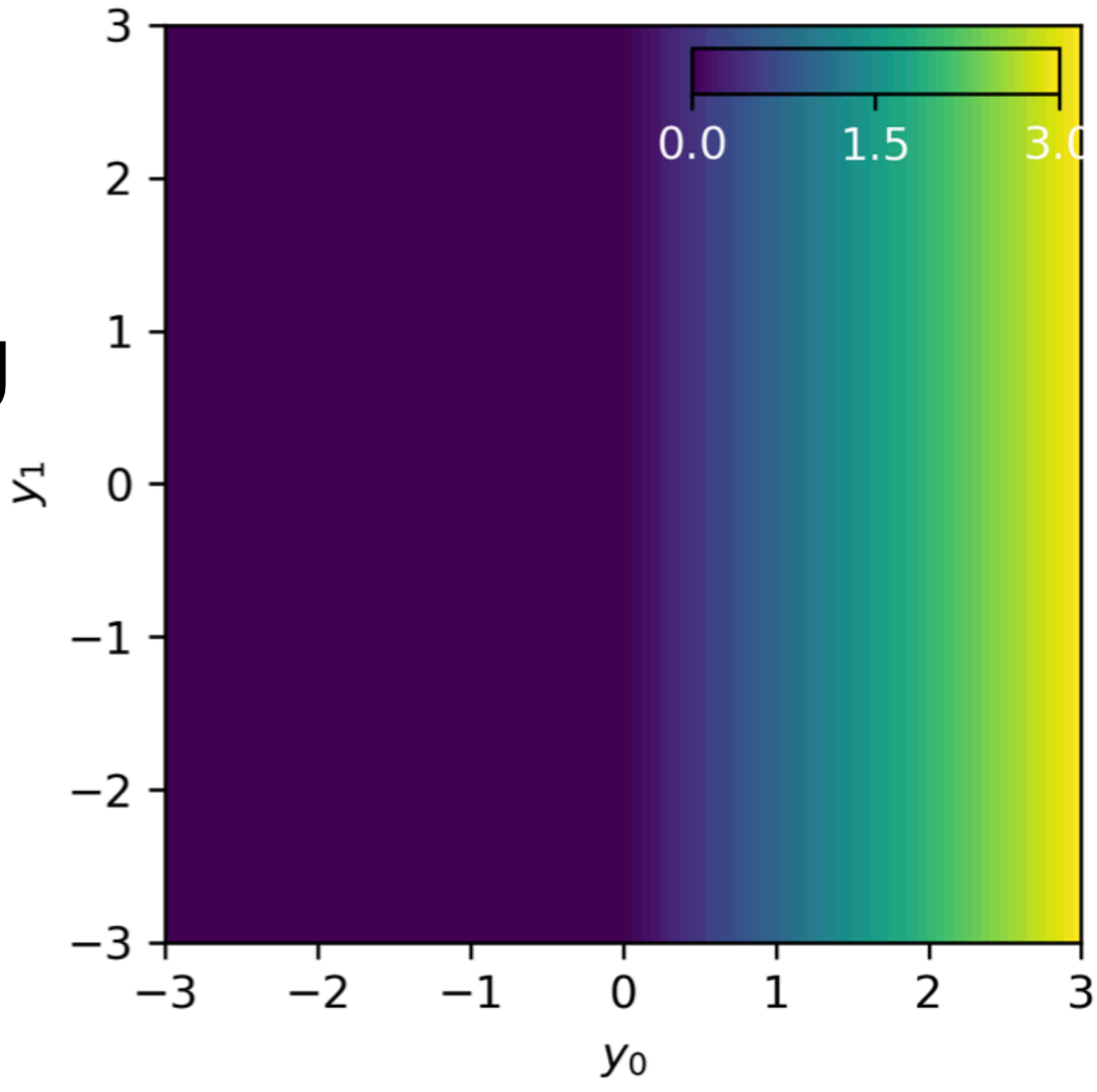



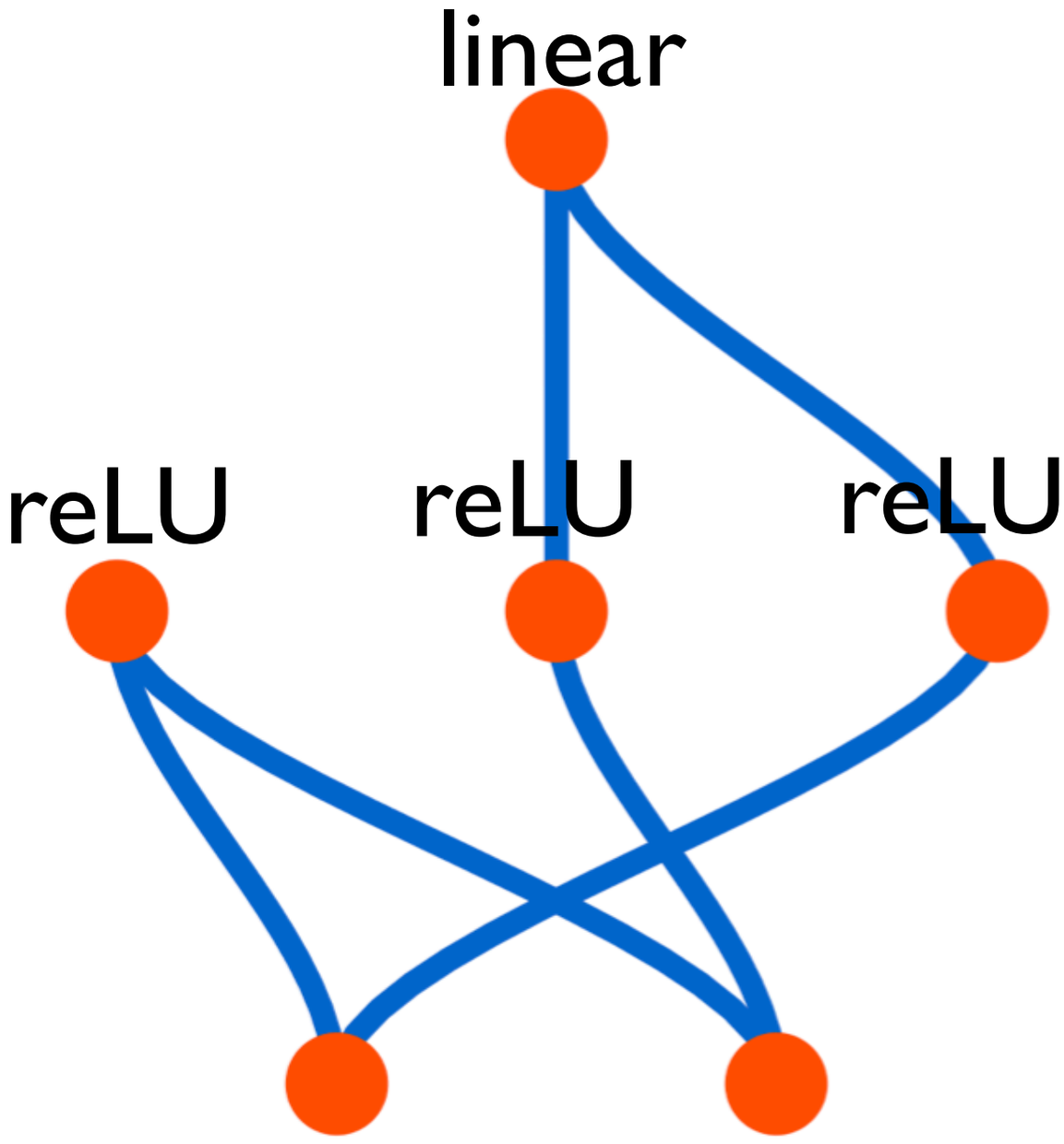
```
# Using the alternative AND function, where the right
# result comes about only for the specific inputs (-1,-1), (-1,+1)
# etc etc

visualize_network(weights=[ [ [-1.0,1.0],
                              [1,-1] ],
                            [ [1,1] ]  ],
                  biases=[ [-1.5,-1.5], [0] ],
                  activations=[ 'jump', 'linear' ],
                  y0range=[-3,3],y1range=[-3,3], M=400)
```

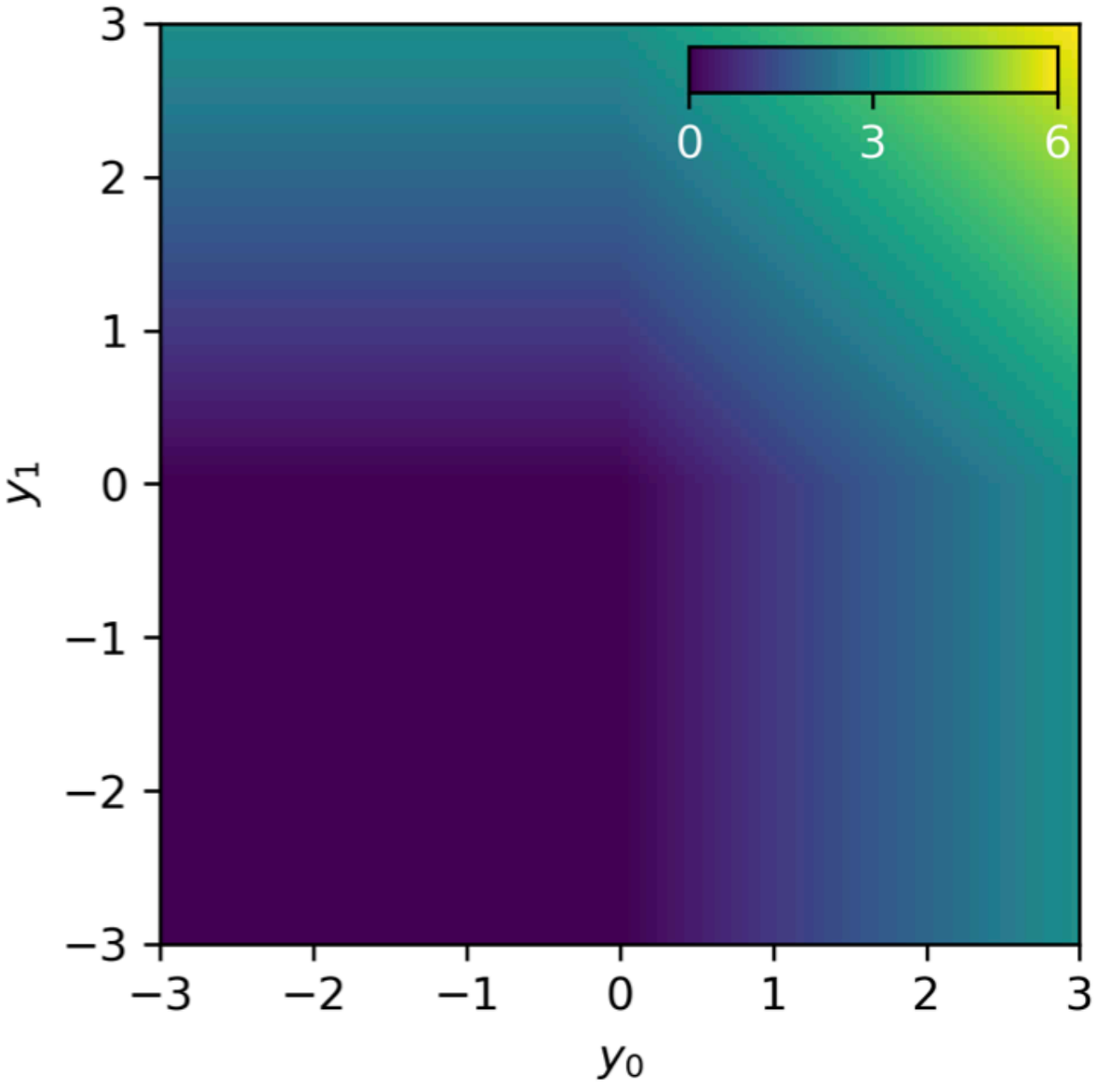


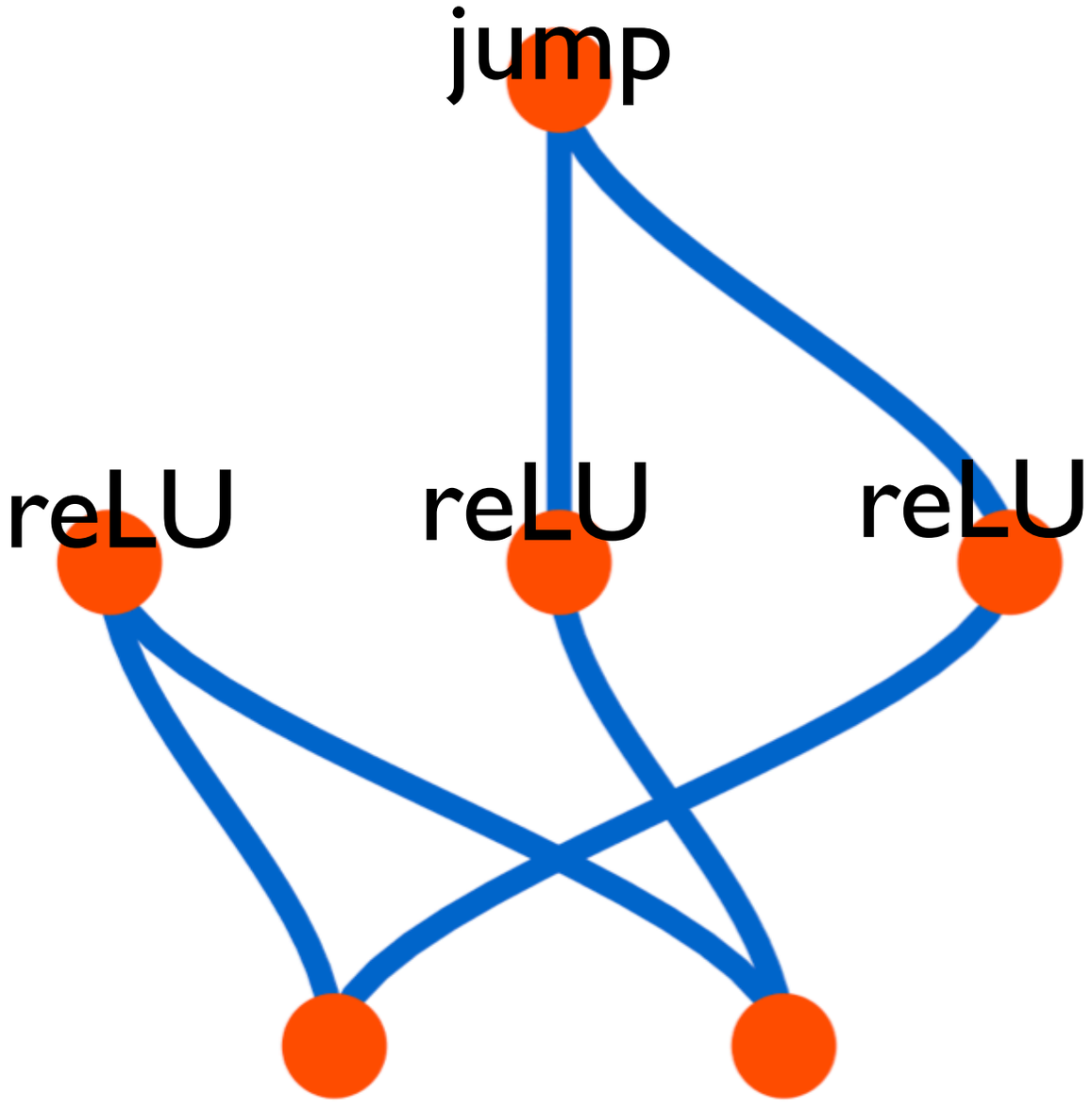


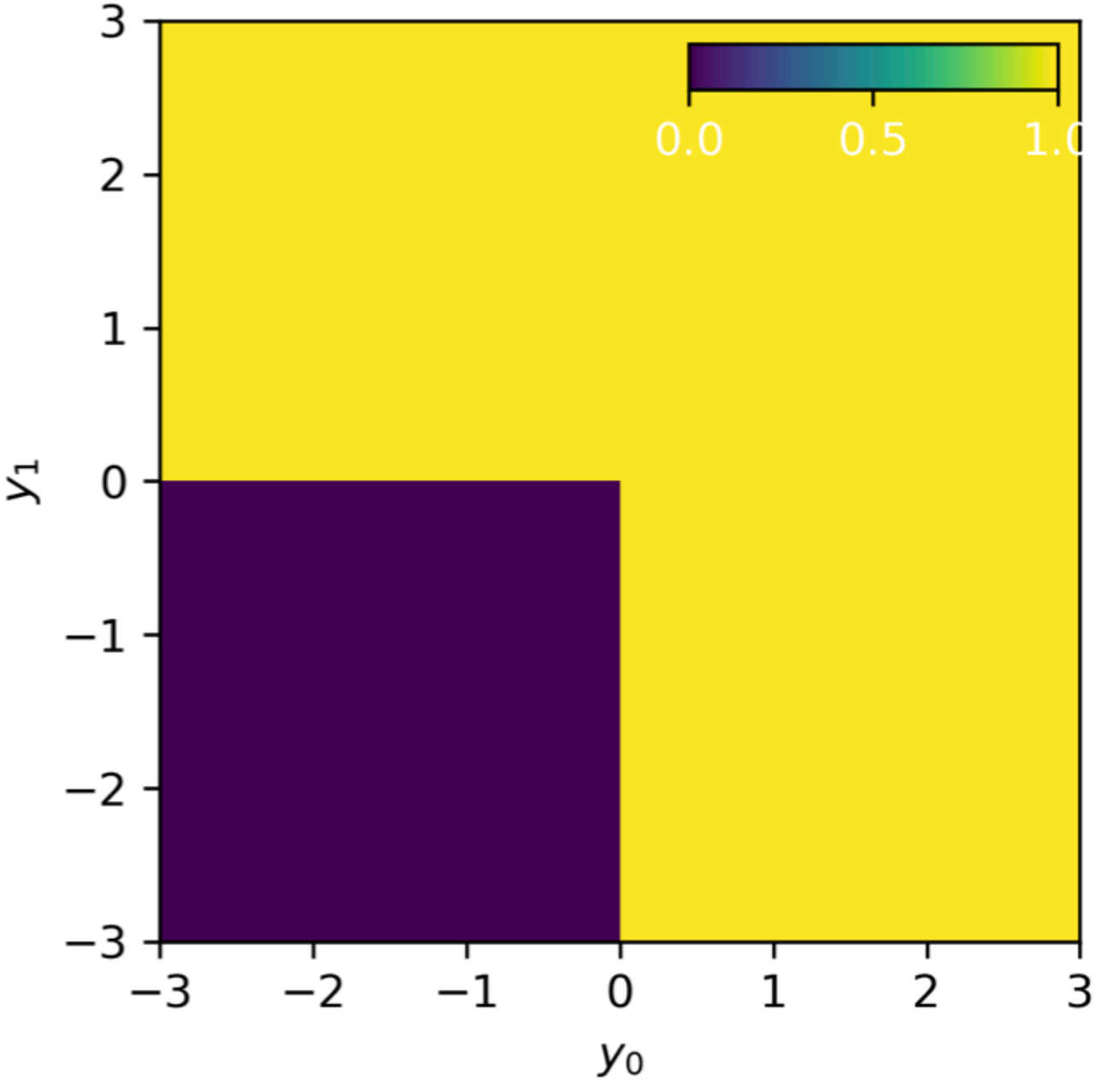

linear
reLU
reLU
reLU

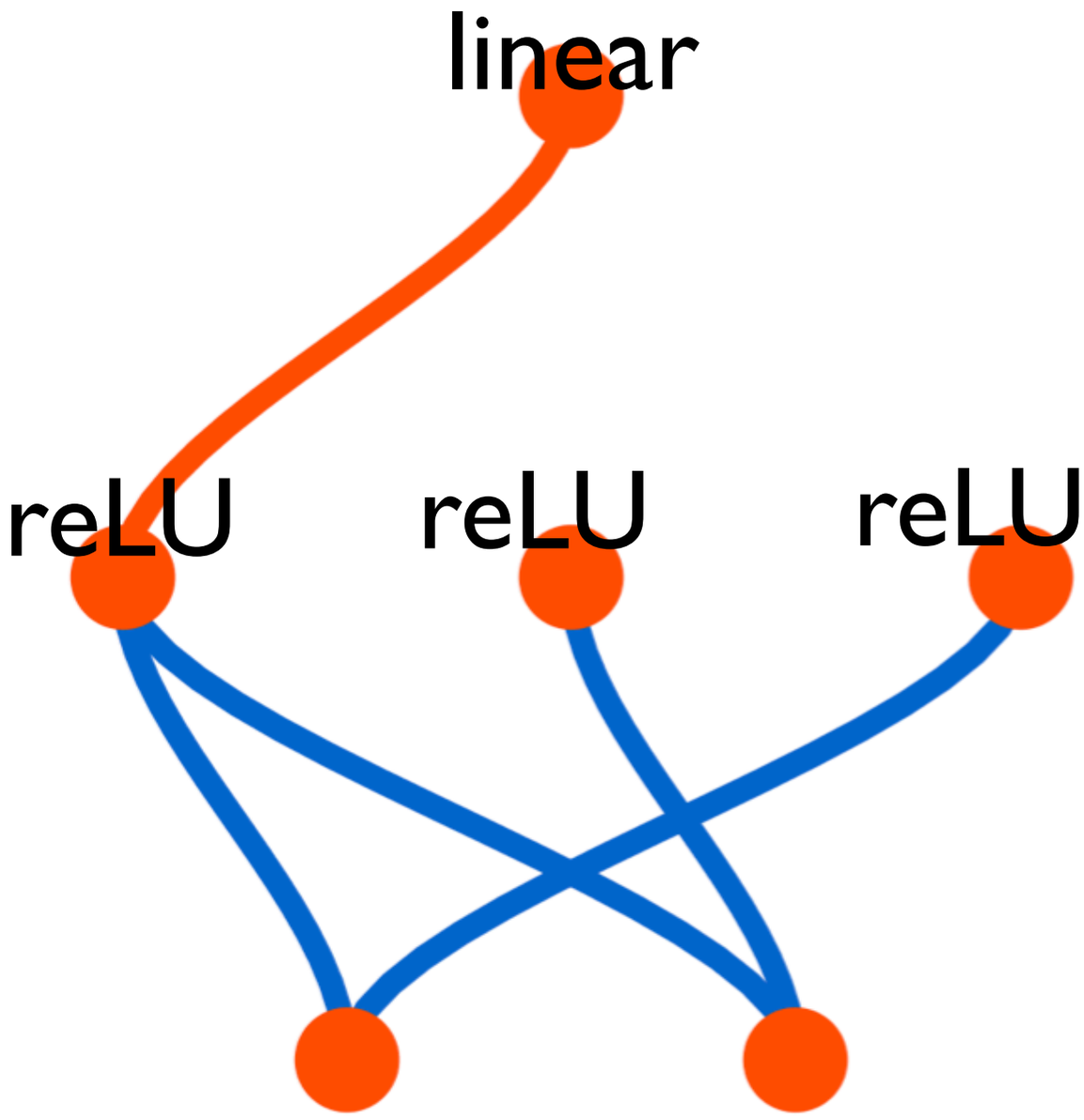

0.0
1.5
3.0
3
2
1
0
−1
−2
−3
$y_1$
−3
−2
−1
0
1
2
3
$y_0$

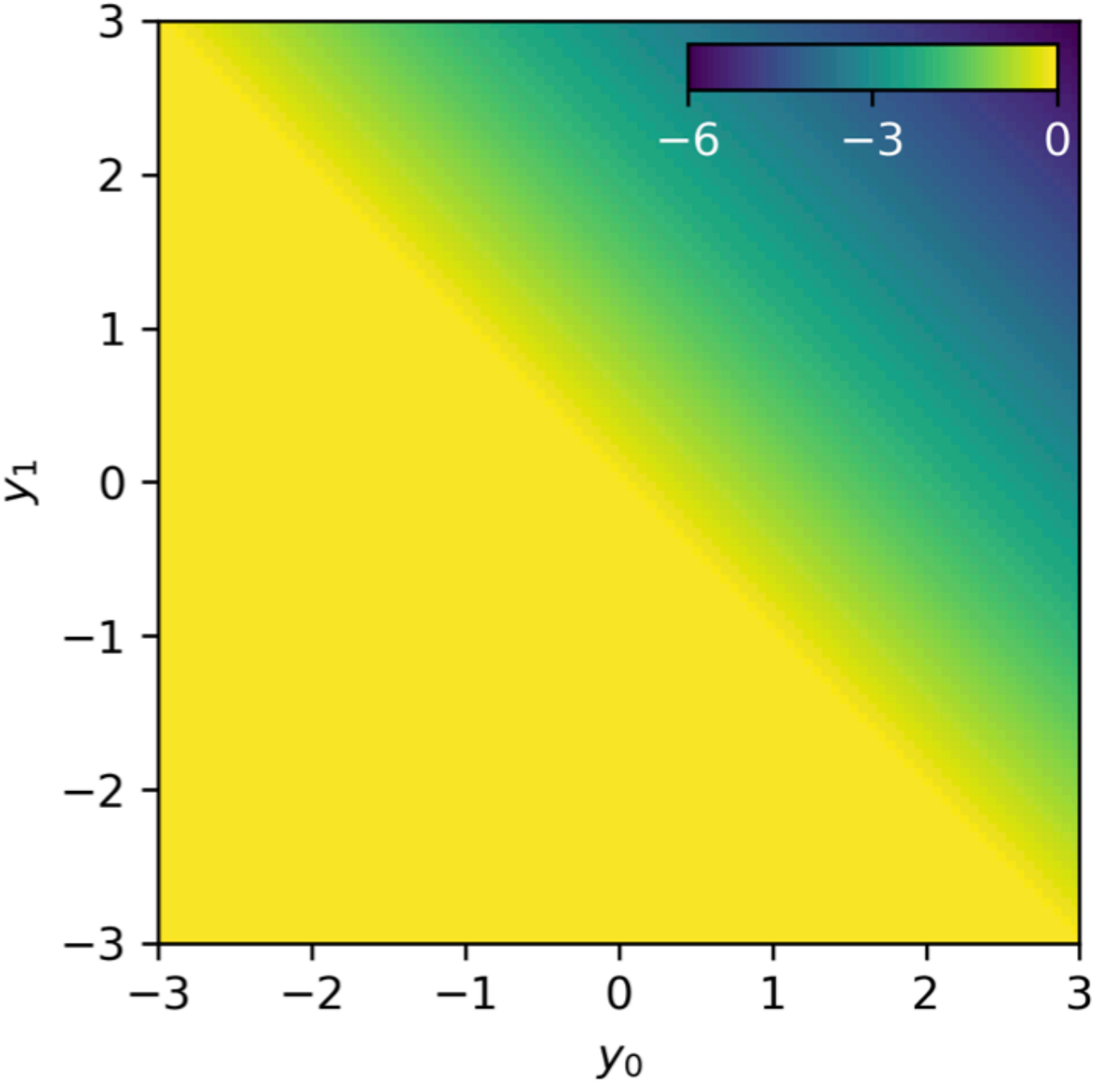

linear
reLU
reLU
reLU

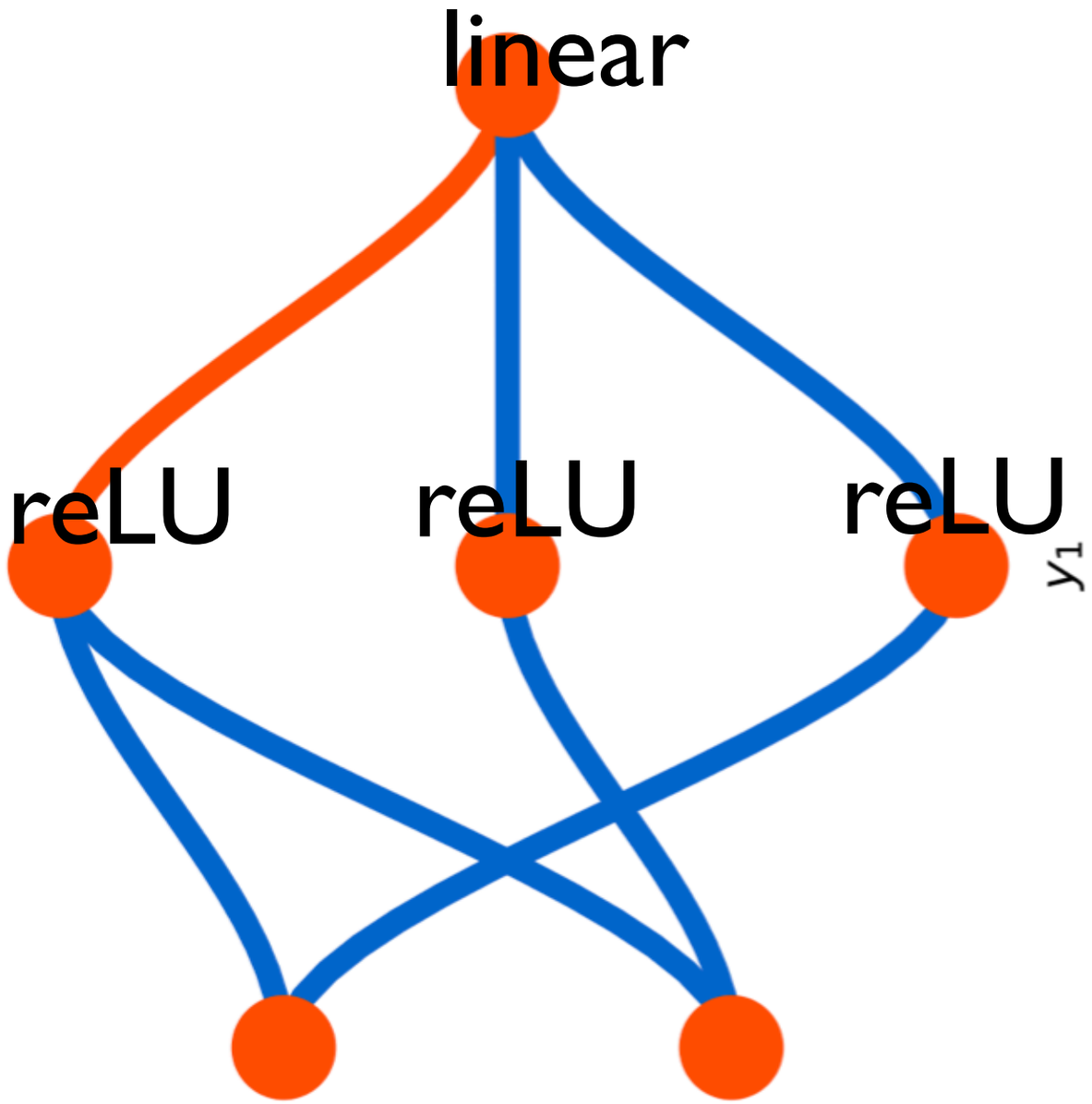

3
2
1
0
−1
−2
−3
$y_1$
0
3
6
−3
−2
−1
0
1
2
3
$y_0$

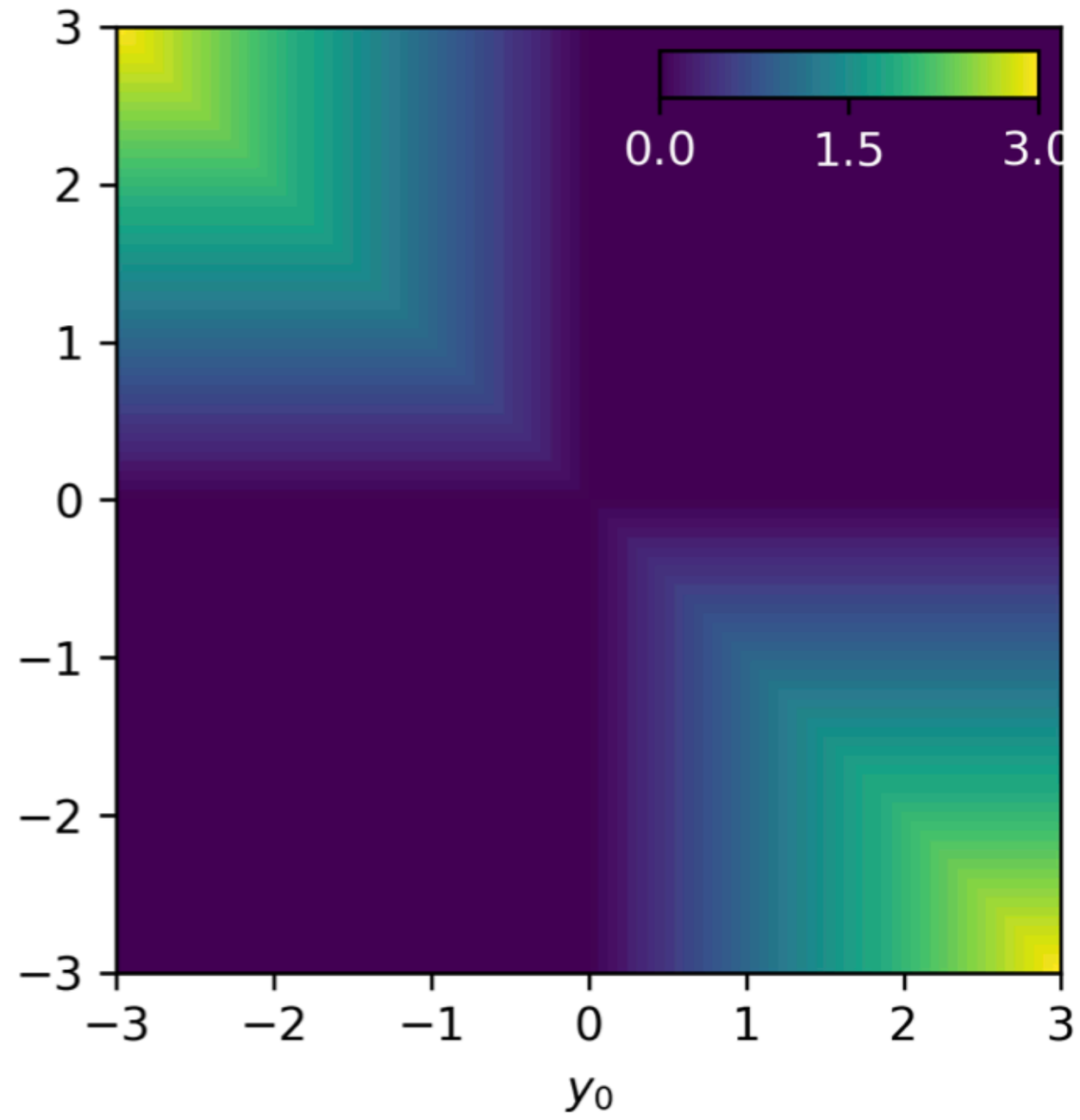

jump
reLU
reLU
reLU

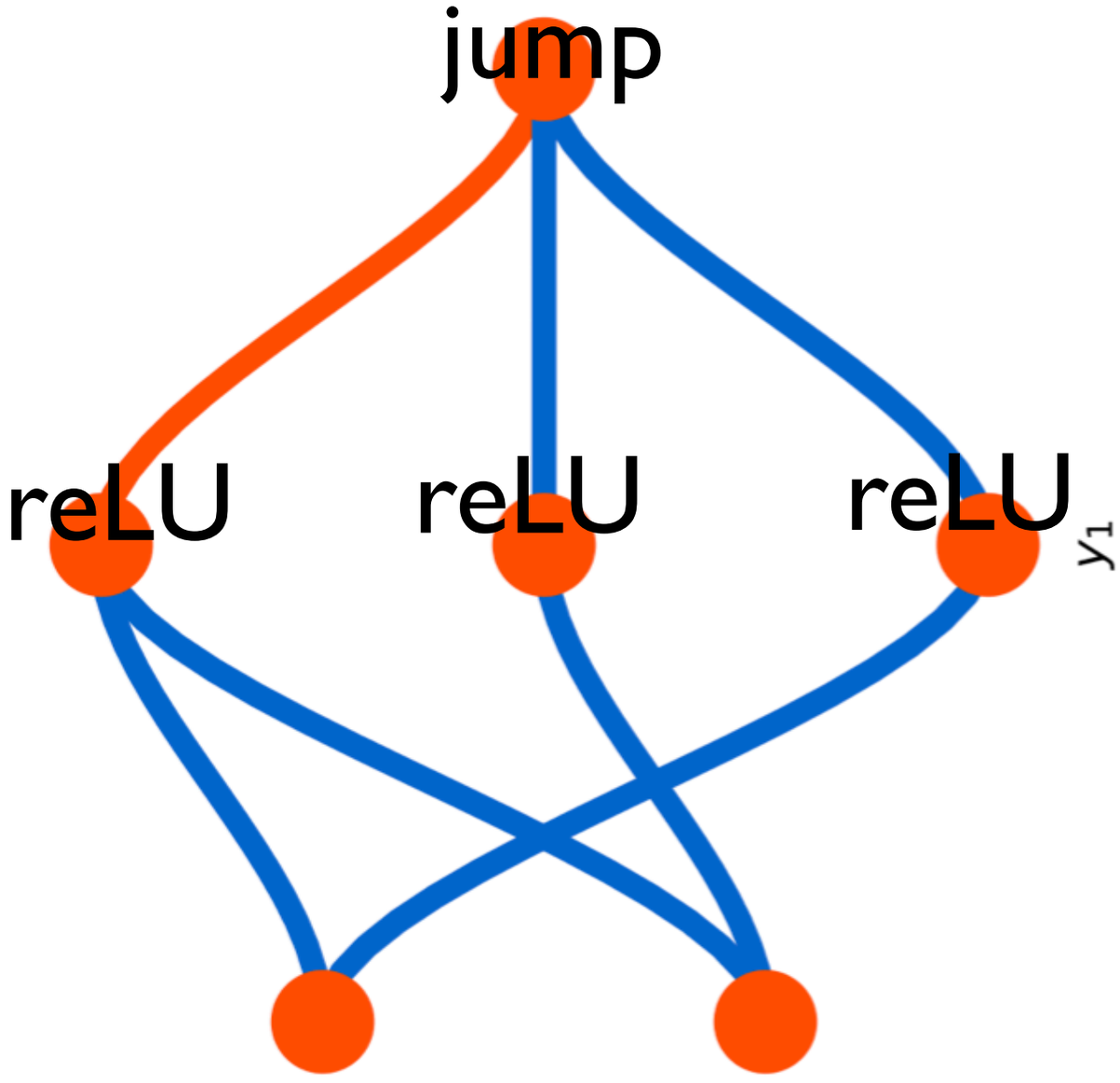

0.0
0.5
1.0
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
$y_0$
$y_1$

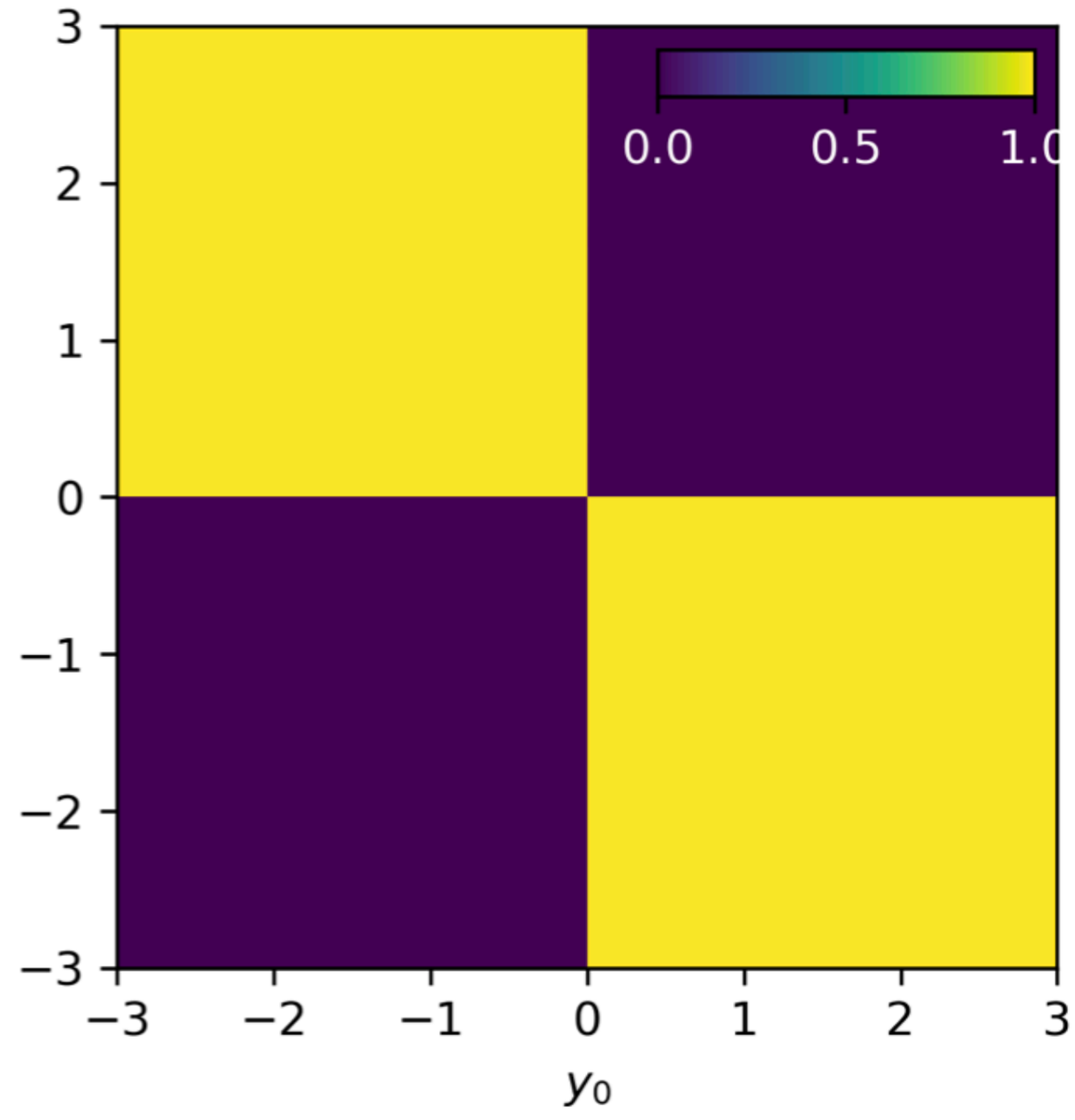

linear
reLU
reLU
reLU


3
2
1
0
−1
−2
−3
−6
−3
0
$y_1$
−3
−2
−1
0
1
2
3
$y_0$


linear
reLU
reLU
reLU

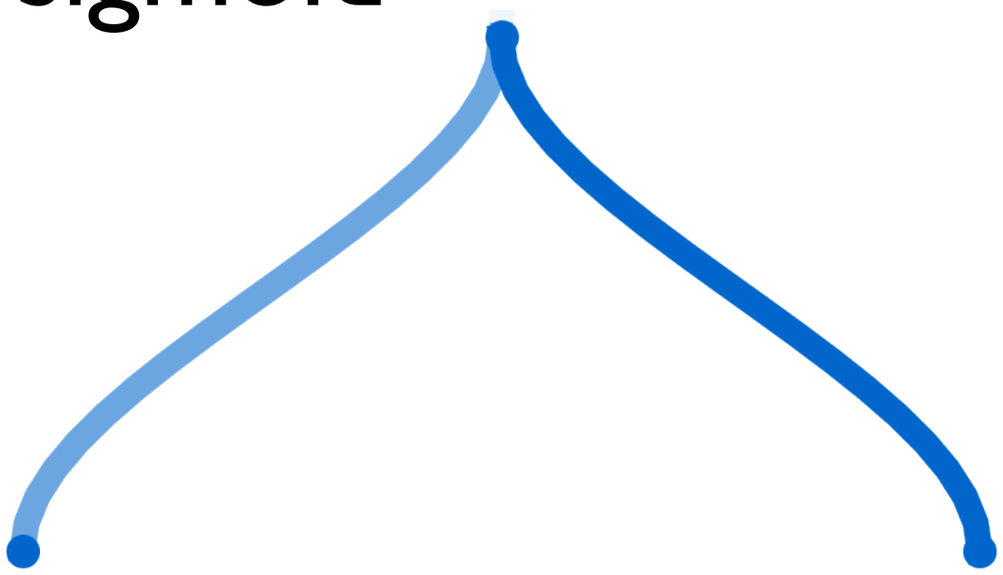

3
2
1
0
−1
−2
−3
$y_1$
0.0
1.5
3.0
−3
−2
−1
0
1
2
3
$y_0$

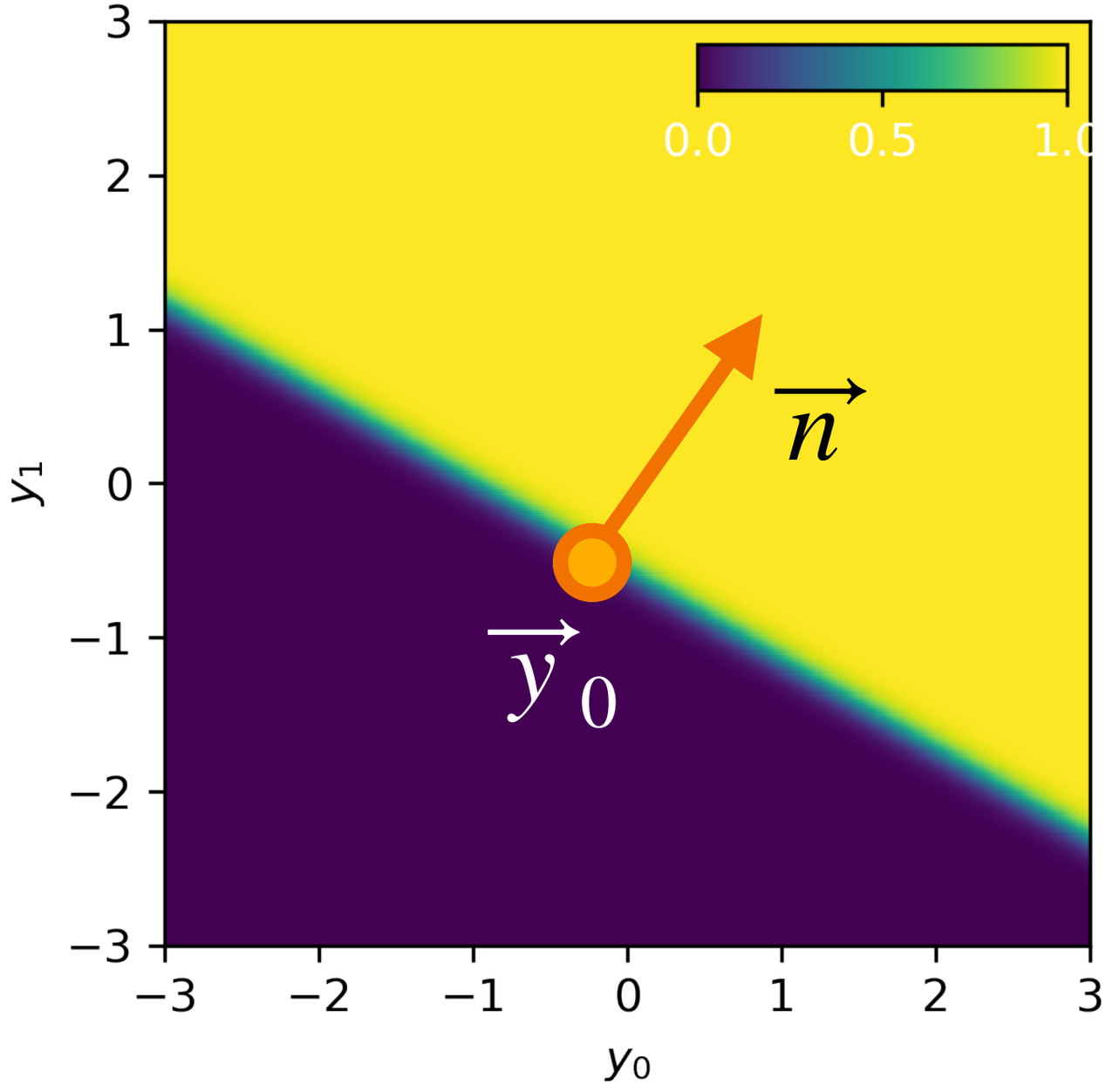

jump
reLU
reLU
reLU


3
2
1
0
−1
−2
−3
$y_1$
0.0
0.5
1.0
−3
−2
−1
0
1
2
3
$y_0$

And now: A general constructive technique (approximate), for arbitrary functions, with one hidden layer!

sigmoid

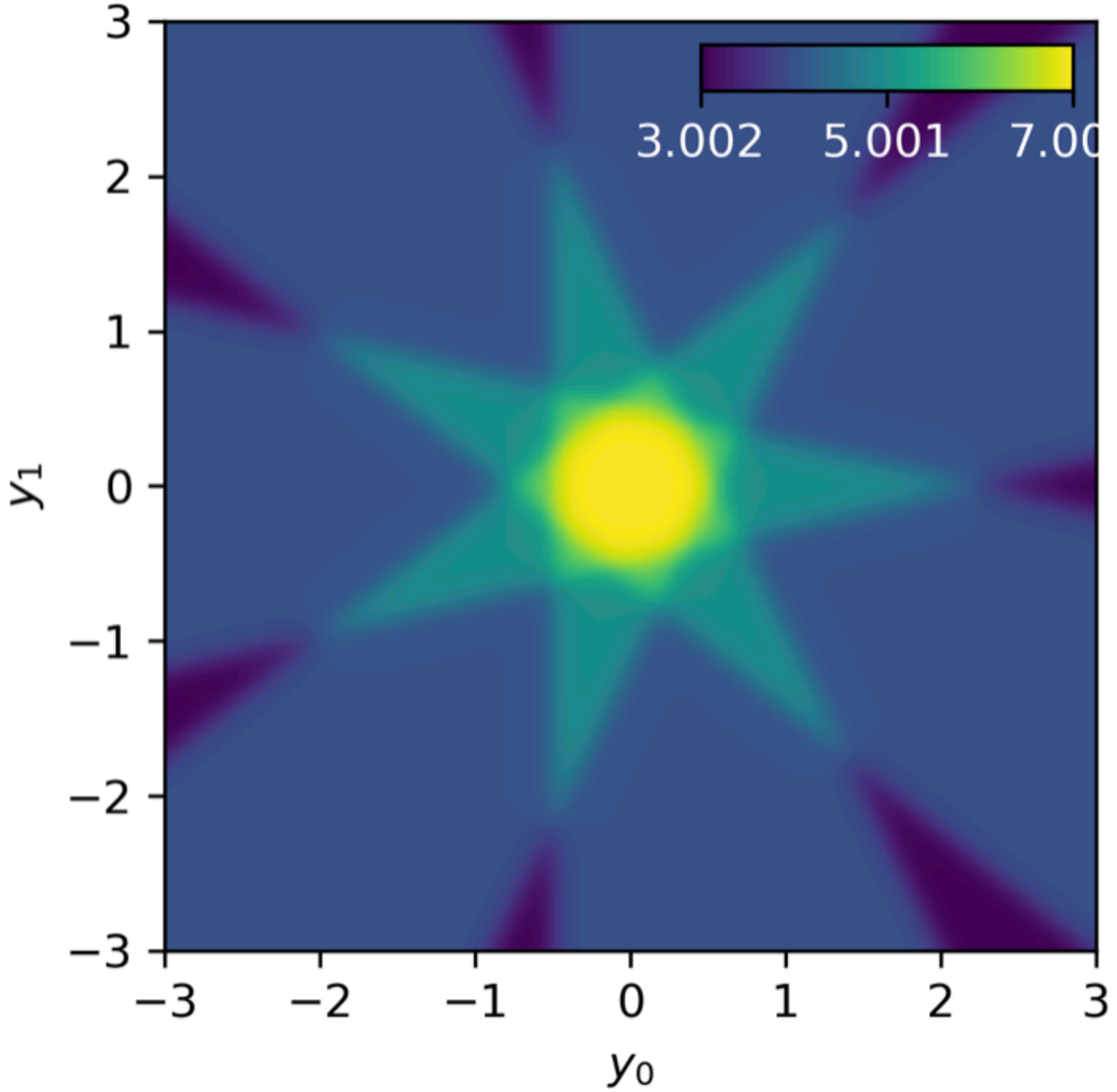





$$y_{\text{out}} = f(\vec{n} \cdot (\vec{y} - \vec{y}_0))$$

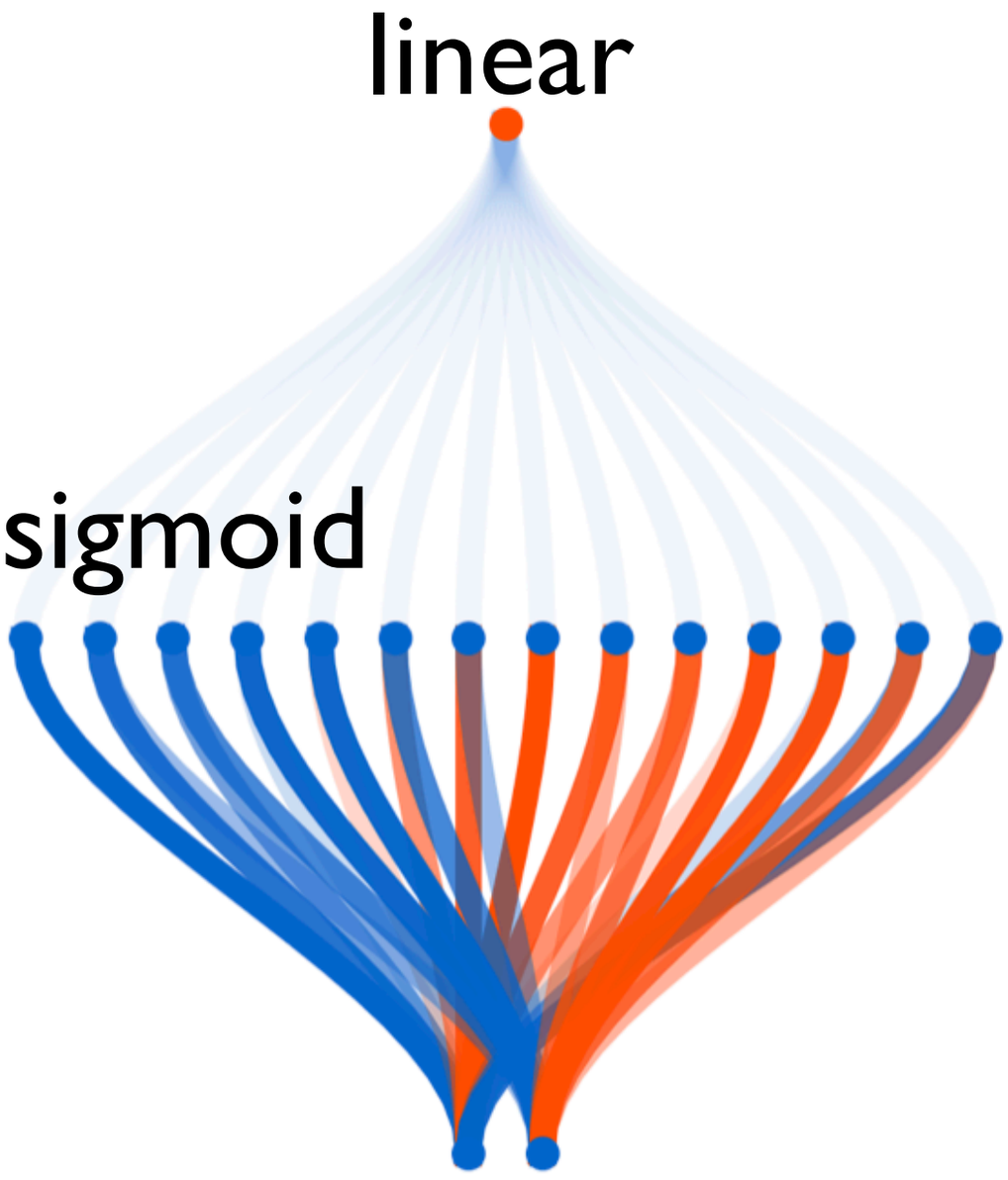
linear

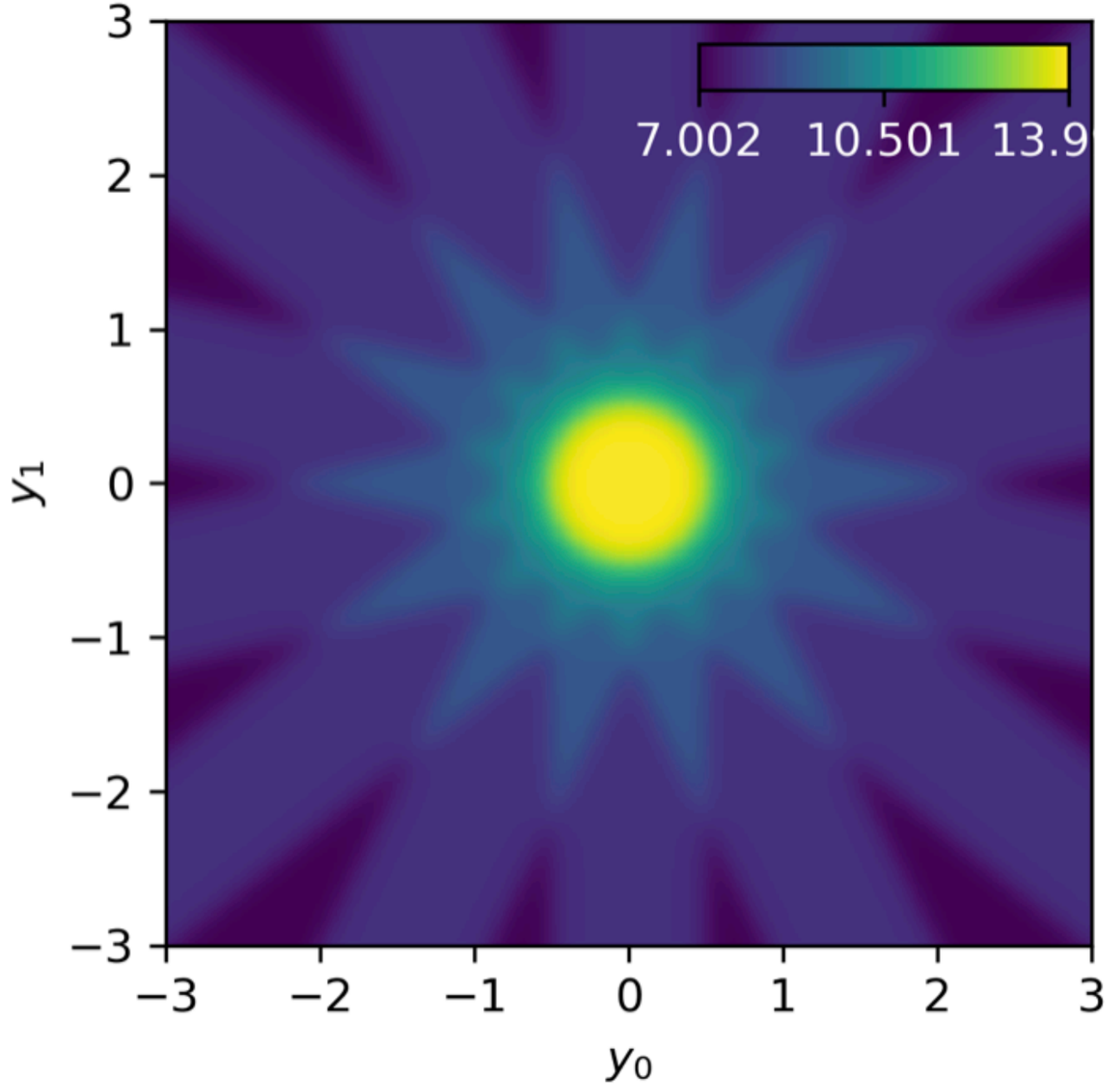
sigmoid

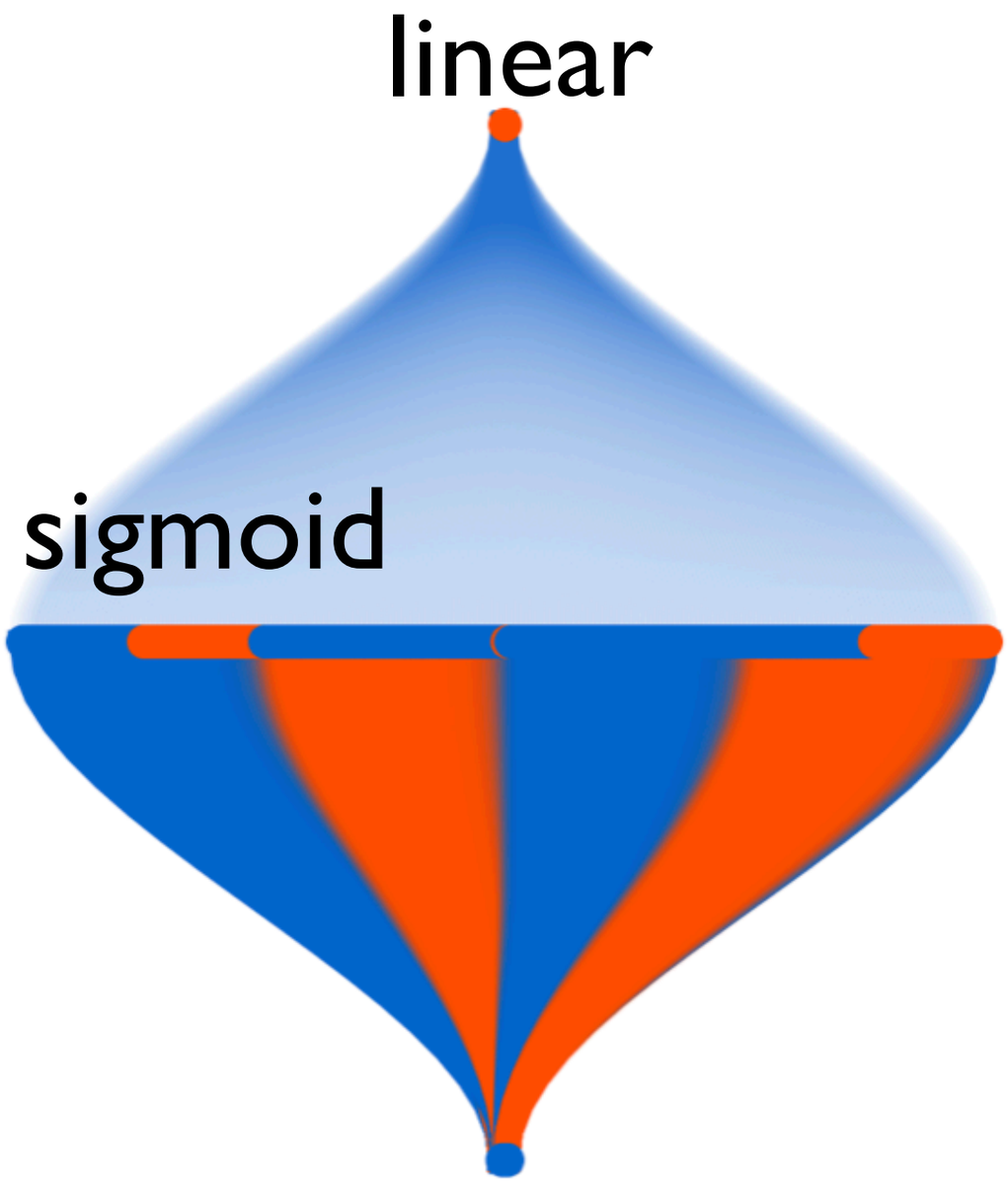


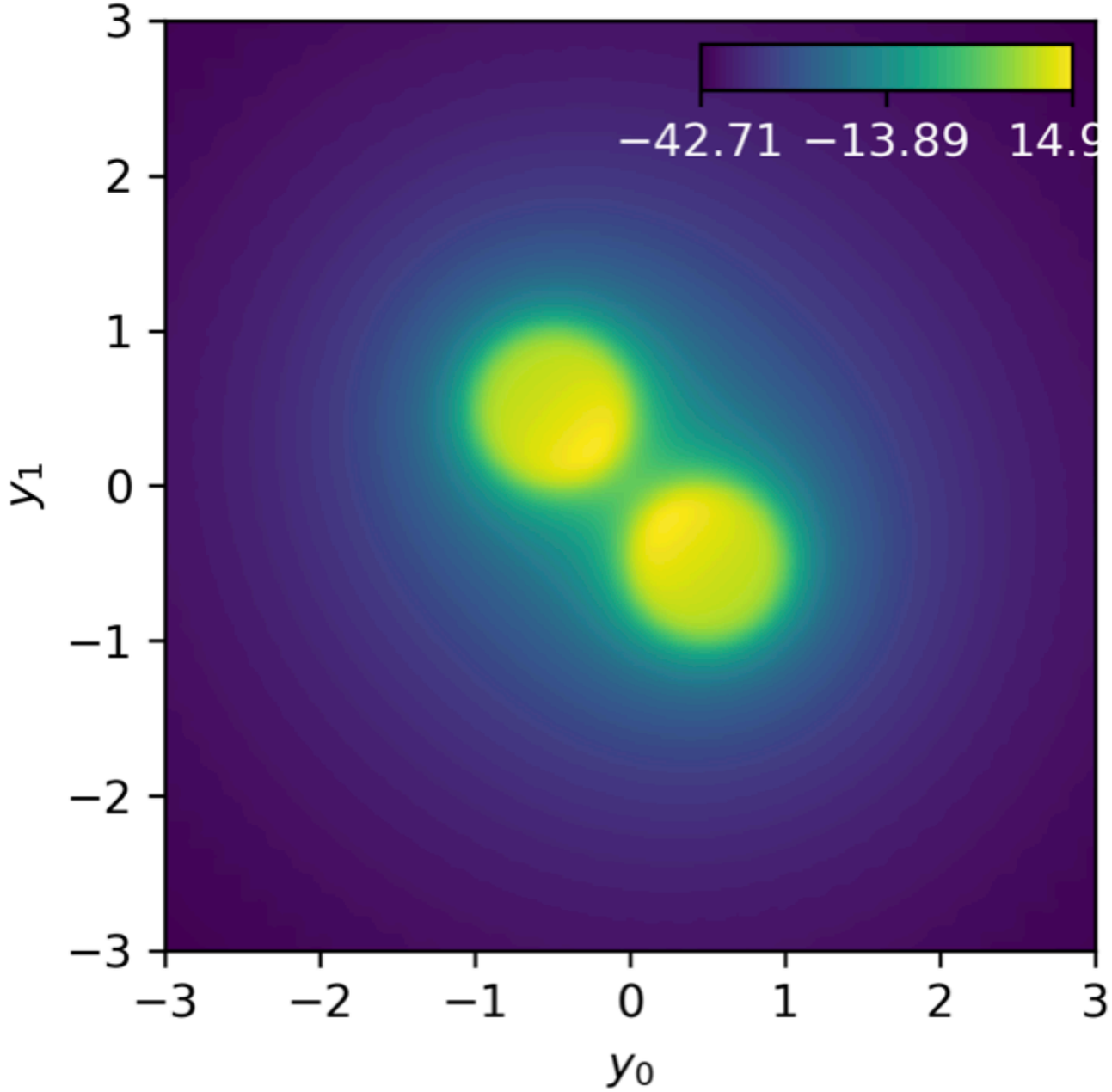
linear

sigmoid



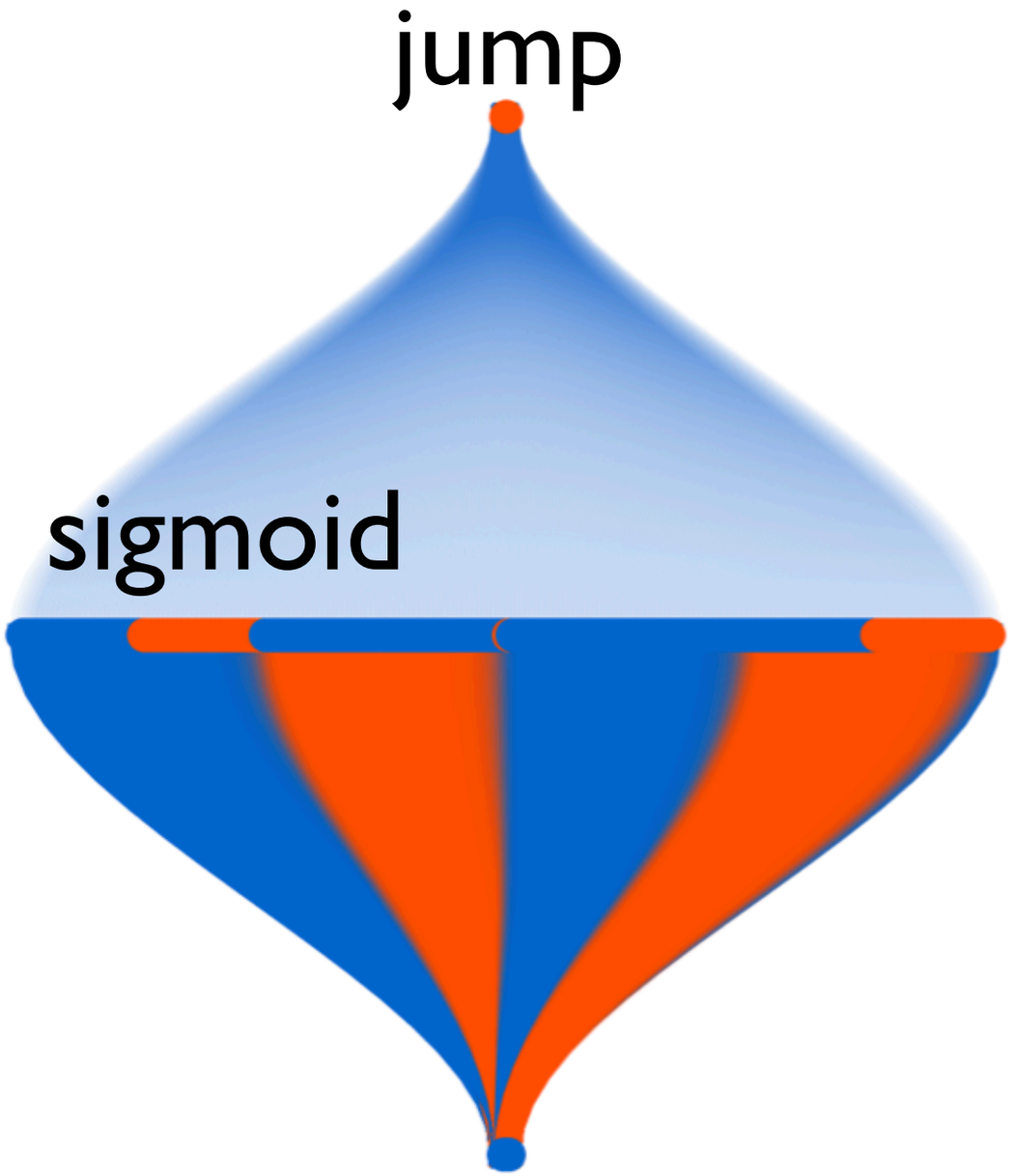


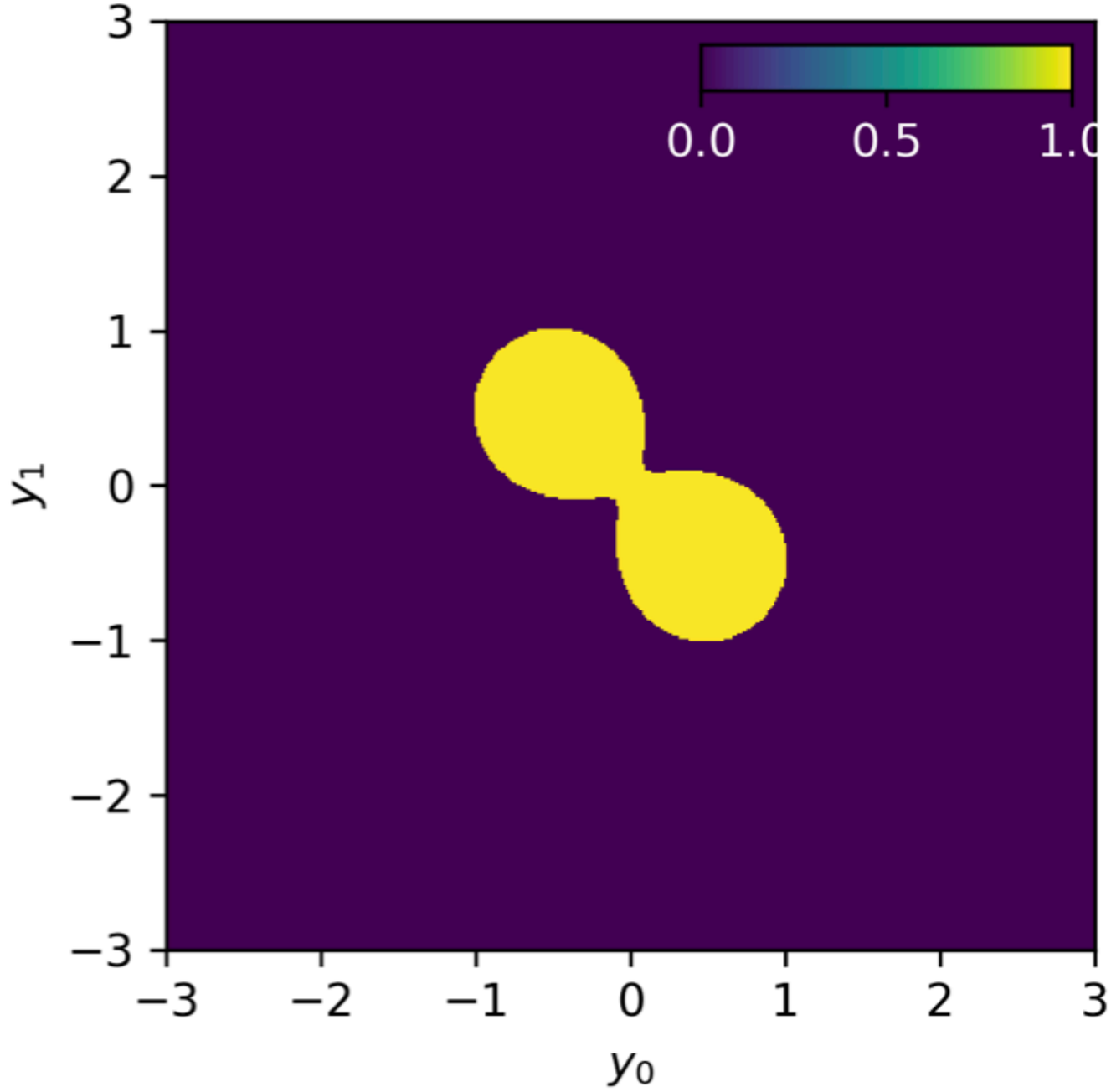



(and we could add more 'blobs')

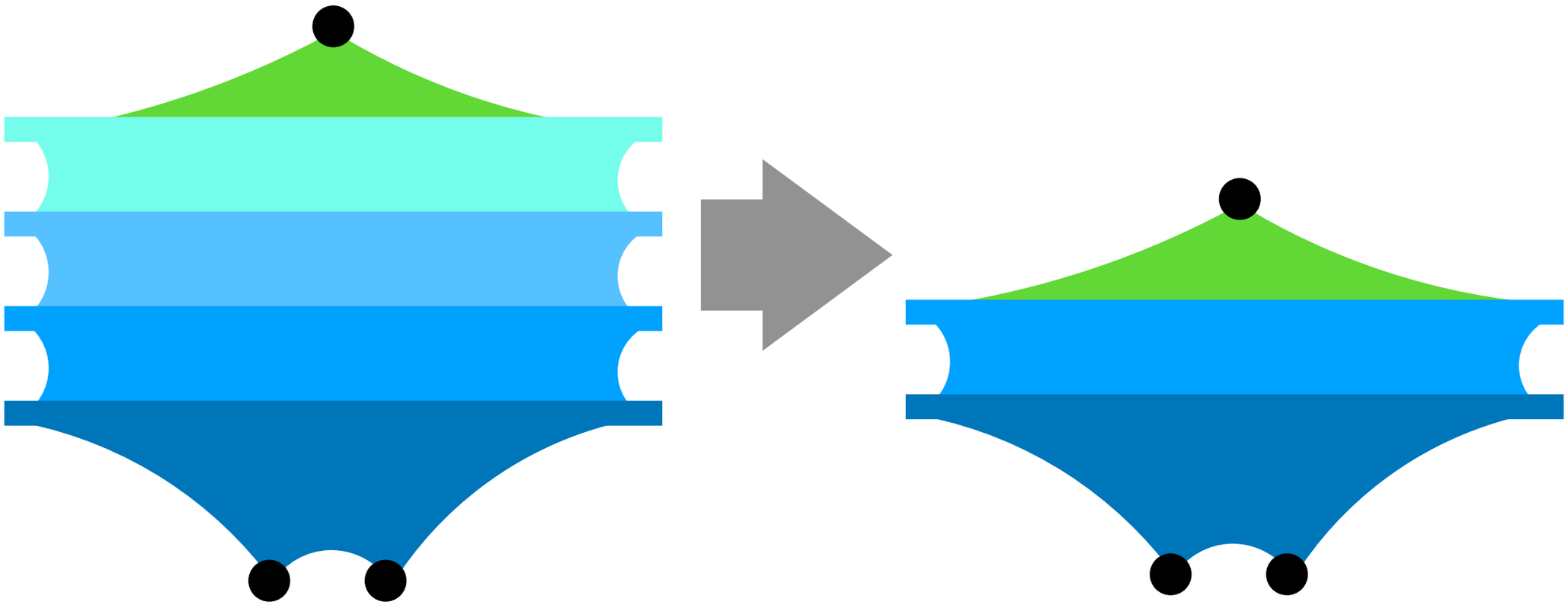
Produce **arbitrary** 2D functions with one hidden layer!


jump
sigmoid


0.0
0.5
1.0
3
2
1
0
−1
−2
−3
$y_1$
−3
−2
−1
0
1
2
3
$y_0$

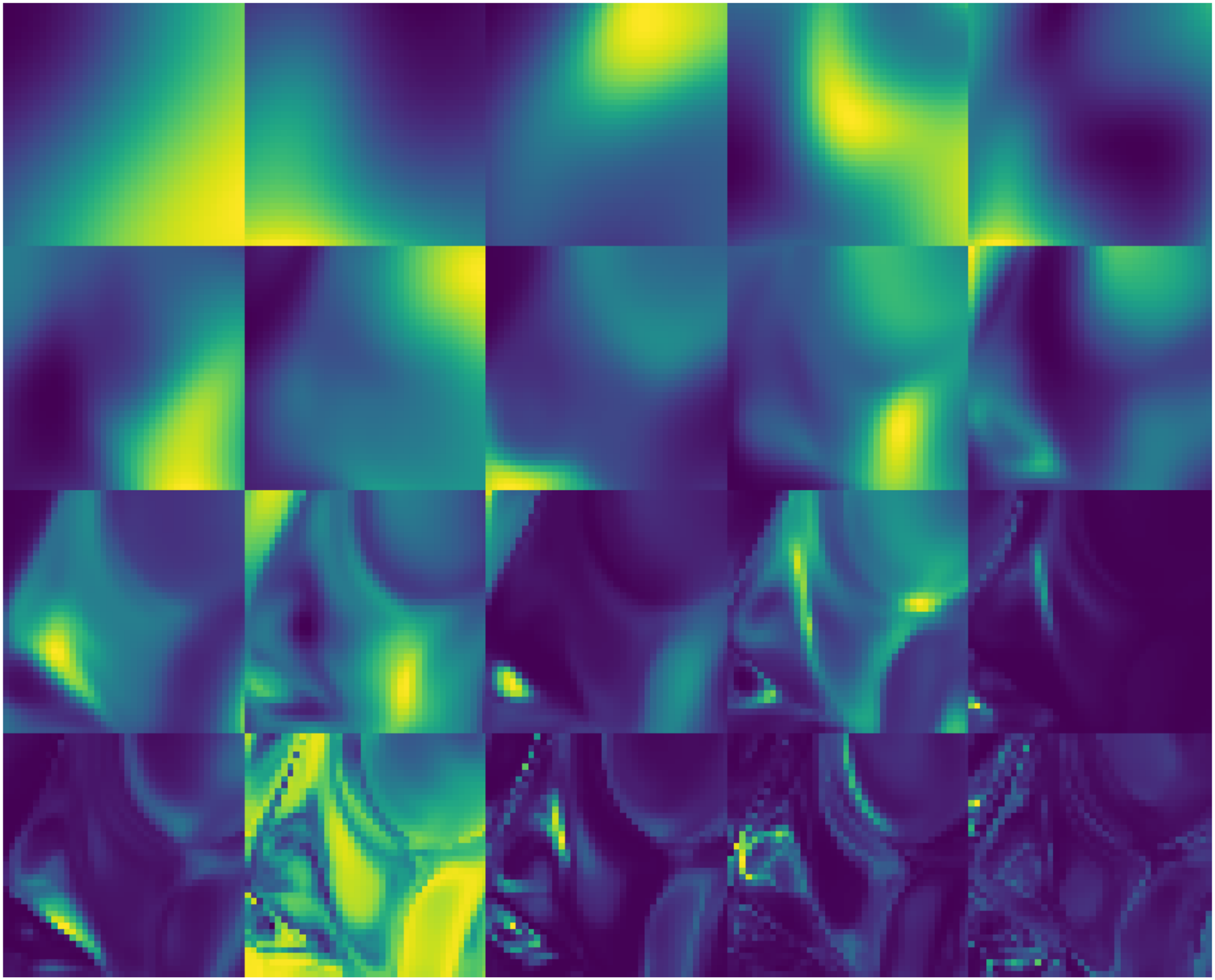
# 3 Visualize the results of intermediate layers in a multi-layer randomly initialized NN

# Visualizing intermediate layers



(cut away higher layers)

# Visualizing intermediate layers

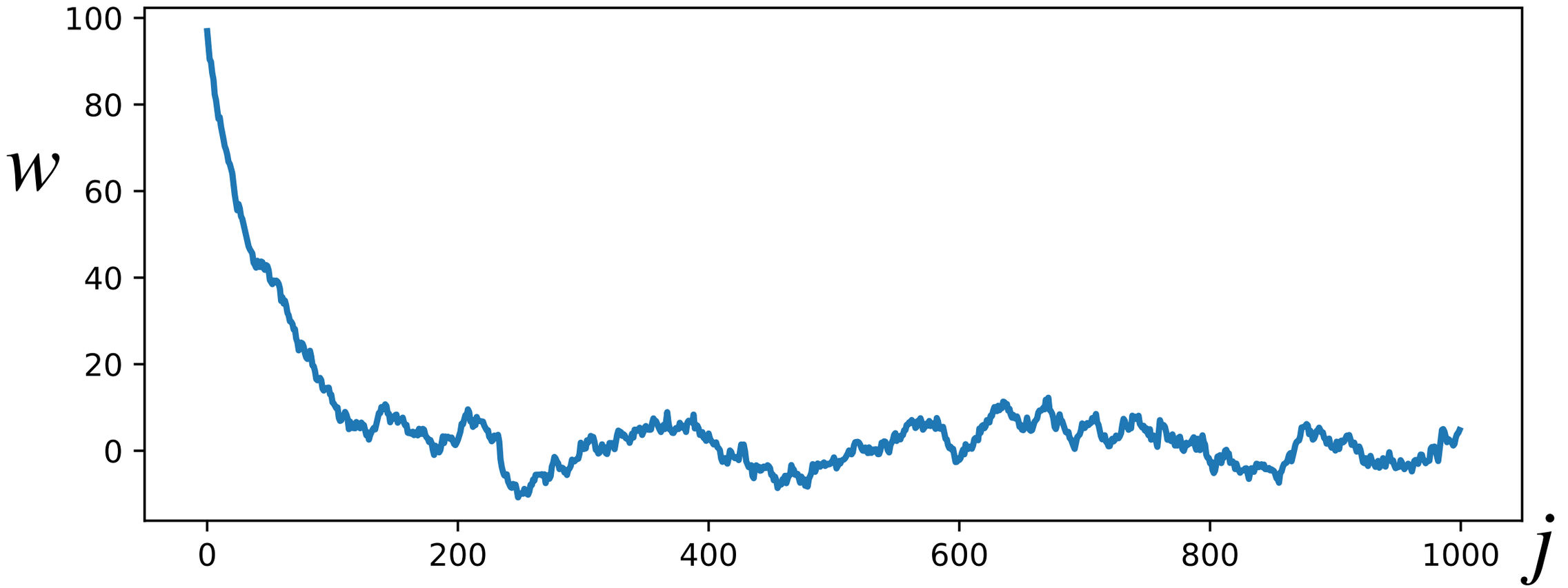


4. What happens when you change the spread of the random weights?
5. Explore cases of curve fitting where there are several (non-equivalent) local minima. Is sampling noise helpful?

# Quiz

# Machine Learning for Physicists

# Stochastic gradient descent




$$w_{j+1} = w_j - \eta w_j + \xi_j \qquad \left\langle \xi_j^2 \right\rangle = 1$$

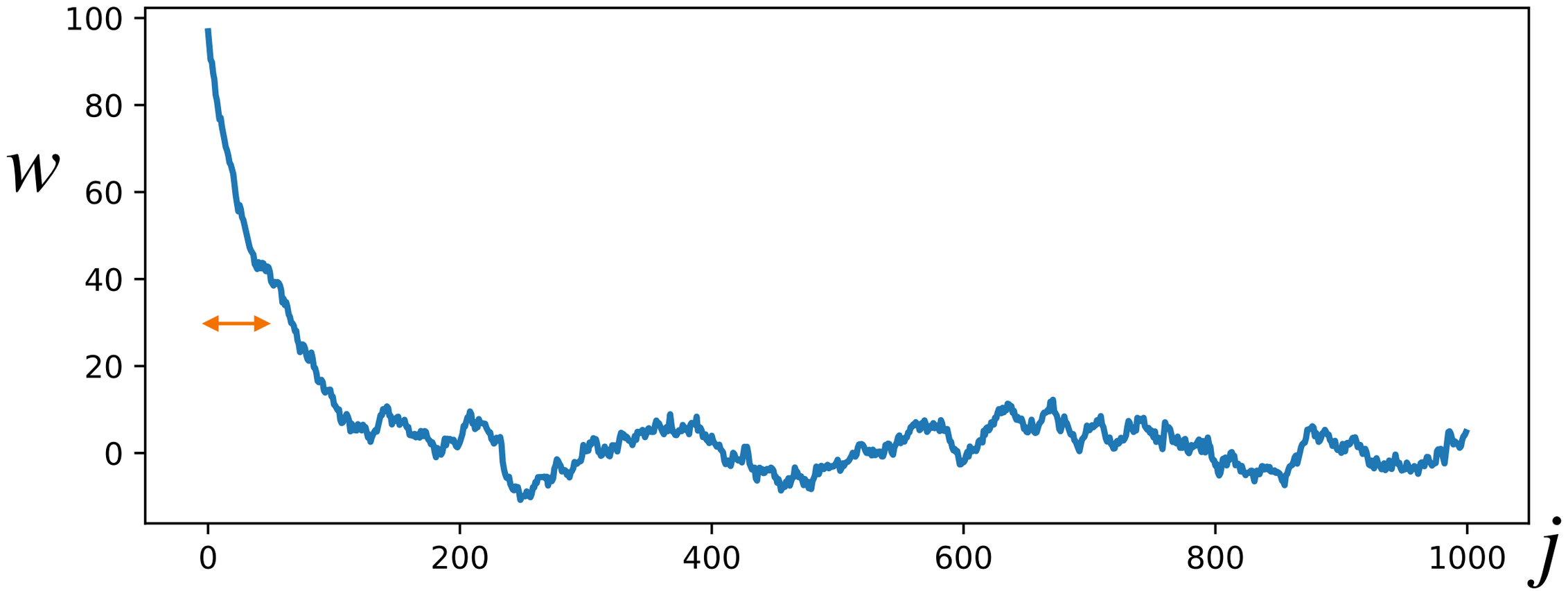
What is the value of $\eta$ ?

**a** 1.0 **d** 0.02

**b** 0.1 **e** 0.01

**c** 0.2 **f** 0.001

# Stochastic gradient descent




$$w_{j+1} = w_j - \eta w_j + \xi_j \qquad \left\langle \xi_j^2 \right\rangle = 1$$

What is the value of $\eta$ ?

**a** 1.0

**b** 0.1

**c** 0.2

**d** 0.02

**e** 0.01

**f** 0.001

# Backpropagation

$$z = f(w_3 f(w_2 f(w_1 y)))$$

$$\frac{\partial z}{\partial w_2} =$$

**a** $f'(w_3 f(w_2 f(w_1 y)))\, f'(w_2 f(w_1 y))\, f(w_1 y)$

**b** $f'(w_3 f(w_2 f(w_1 y)))\, w_3 f'(w_2 f(w_1 y))\, w_2 f(w_1 y)$

**c** $f'(w_3 f(w_2 f(w_1 y)))\, w_3 f'(w_2 f(w_1 y))\, f(w_1 y)$

**d** $f'(w_3 f(w_2 f(w_1 y)))\, w_3 f'(w_2 f(w_1 y))\, f'(w_1 y)$

# Backpropagation

$$z = f(w_3 f(w_2 f(w_1 y)))$$

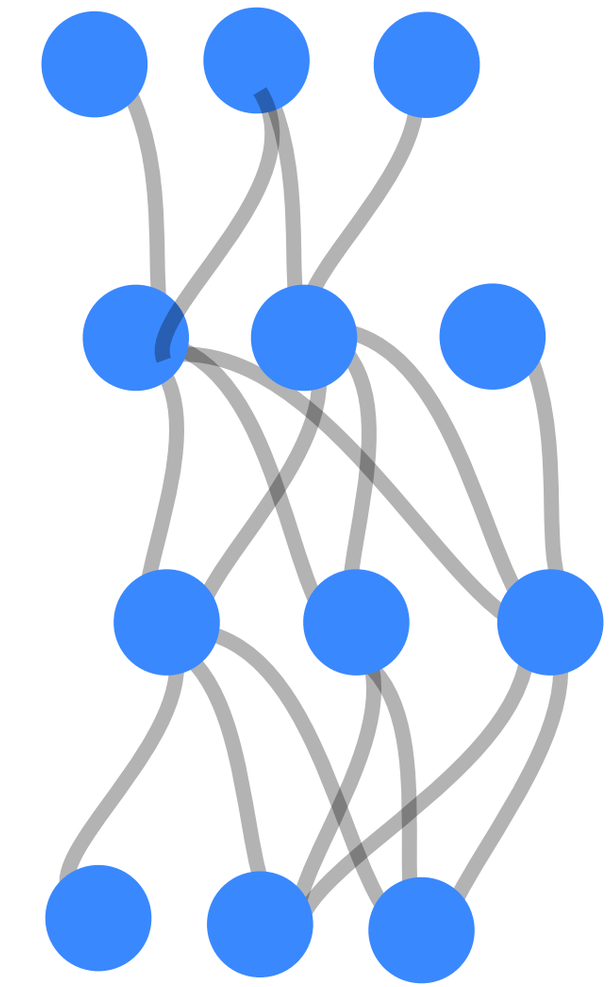
$$\frac{\partial z}{\partial w_2} =$$

**a** $f'(w_3 f(w_2 f(w_1 y)))\, f'(w_2 f(w_1 y))\, f(w_1 y)$

**b** $f'(w_3 f(w_2 f(w_1 y)))\, w_3 f'(w_2 f(w_1 y))$

**c** $f'(w_3 f(w_2 f(w_1 y)))\, w_3 f'(w_2 f(w_1 y))\, f(w_1 y)$

**d** $f'(w_3 f(w_2 f(w_1 y)))\, w_3 f'(w_2 f(w_1 y))\, f'(w_1 y)$



here N=4

# Backpropagation #2

In a network with N layers of M neurons each, and for S samples:

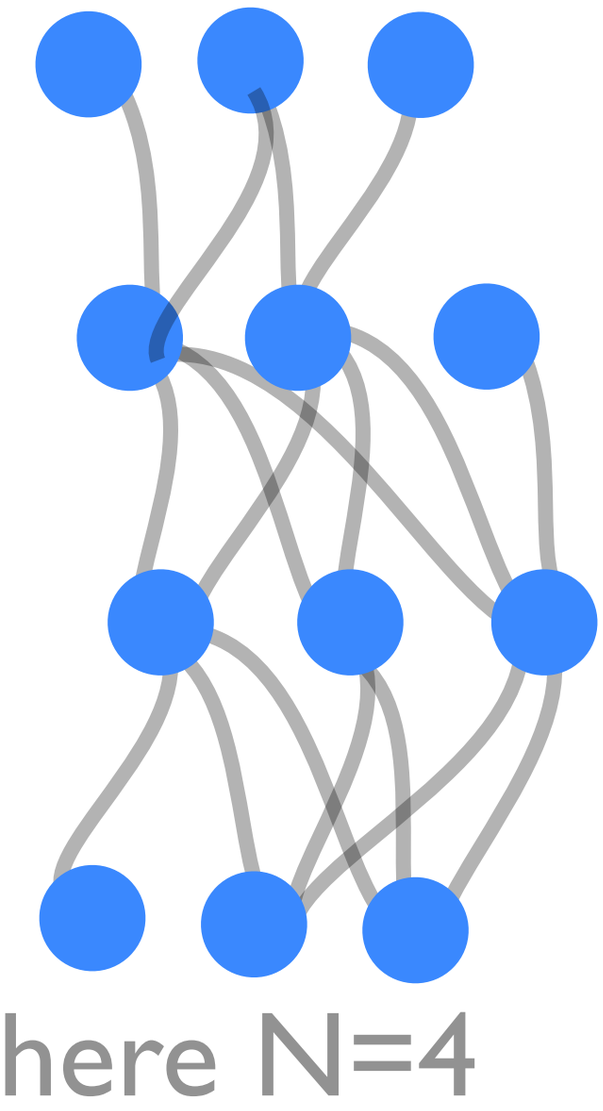
What would be the amount of network evaluations required if one tries to deduce all the derivatives of the (sample-averaged) cost function with respect to the weights brute-force numerically?

**a** $(1 + (N-1)M)$

**b** $S(1 + (N-1)M^2)$

**c** $S(1 + (N-1)M)$

**d** $S(1 + (N-1)^2M)$



here N=4

# Backpropagation #2

In a network with N layers of M neurons each, and for S samples:

What would be the amount of network evaluations required if one tries to deduce all the derivatives of the (sample-averaged) cost function with respect to the weights brute-force numerically?

**a** $(1 + (N-1)M)$

**b** $S(1 + (N-1)M^2)$

**c** $S(1 + (N-1)M)$

**d** $S(1 + (N-1)^2 M)$

# Backpropagation #3

```
shape(y_layer[-3-j])==[100,25]
shape(delta)==[100,50]
```

```
shape of:
dot(transpose(y_layer[-3-j]),delta)
```

**a** `[50,25]`

**b** `[25,50]`

**c** `[100,25,100,50]`

**d** `[25,100]`

**e** `[25,100,100,50]`

**f** `[25,100,50]`

# Backpropagation #3

```
shape(y_layer[-3-j])==[100,25]
shape(delta)==[100,50]
```

**shape of:**

```
dot(transpose(y_layer[-3-j]),delta)
```

**a** [50,25]

**b** [25,50]

**c** [100,25,100,50]

**d** [25,100]

**e** [25,100,100,50]

**f** [25,100,50]

# Practice session: training neural networks

Machine Learning for Physicists

**Notebook for online tutorials:**

Tutorial: Network Training Visualization notebook (as pure python script)

This notebook visualizes training of multilayer neural networks!

# Lecture 2 Homework

(will be briefly discussed in the online session for lecture 3)

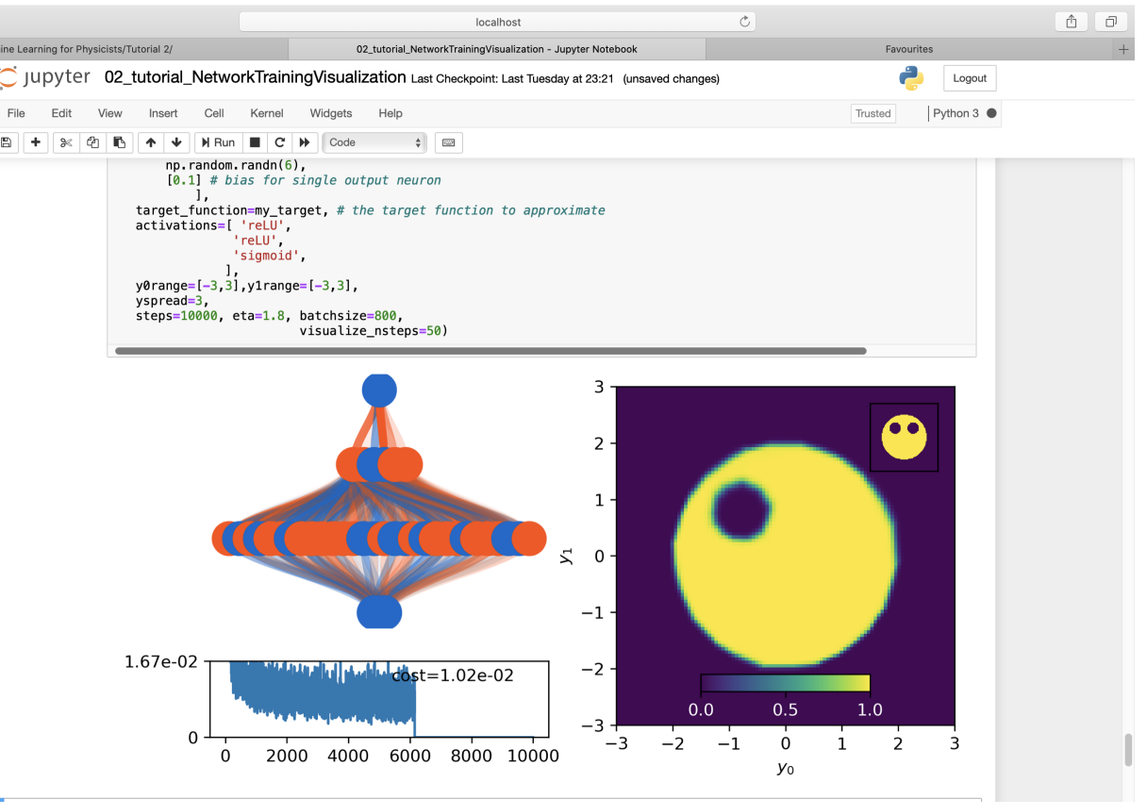
Machine Learning for Physicists

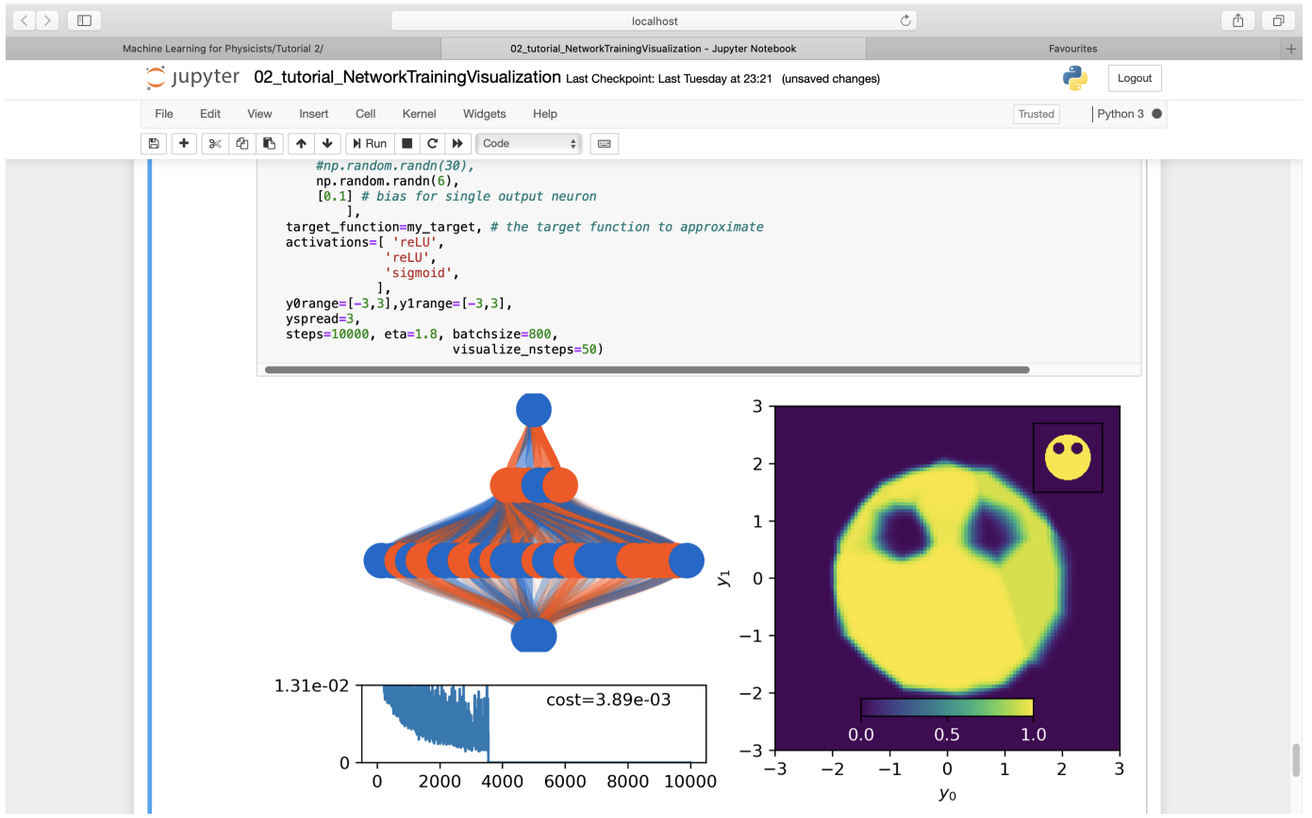
1* Carefully study the backpropagation algorithm, on paper and in the program

2* Visualize the training of a multi-layer network for some interesting function! Explore reLU vs sigmoid!

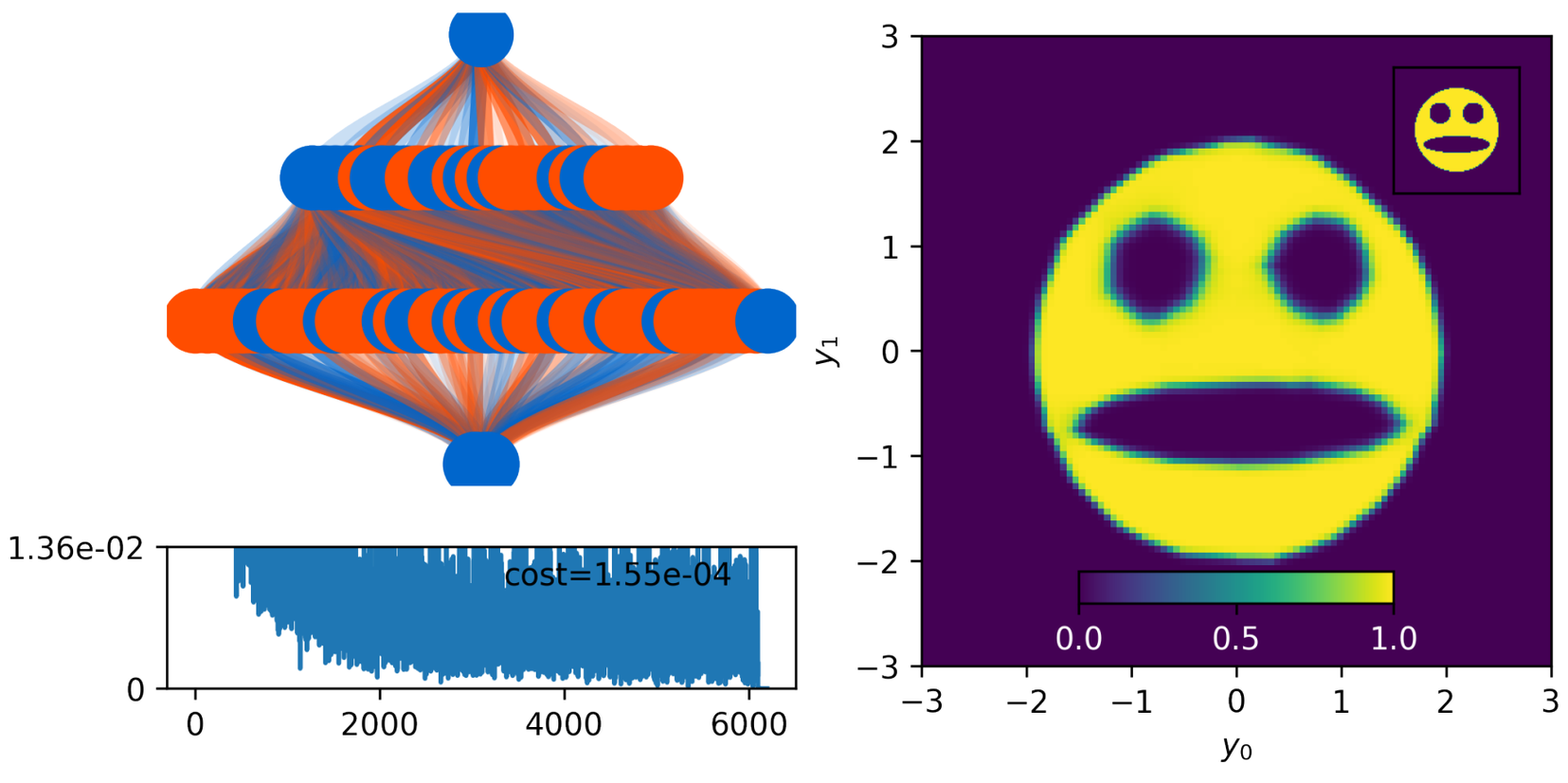
* minimum

3 Analyze the evolution of the slope w during stochastic gradient descent on a cost function given by $C = \frac{1}{2}\sum_{j=1}^{N}(wx_j - \tilde{w}x_j)^2$, where $x_j$ are the samples drawn from a Gaussian distribution



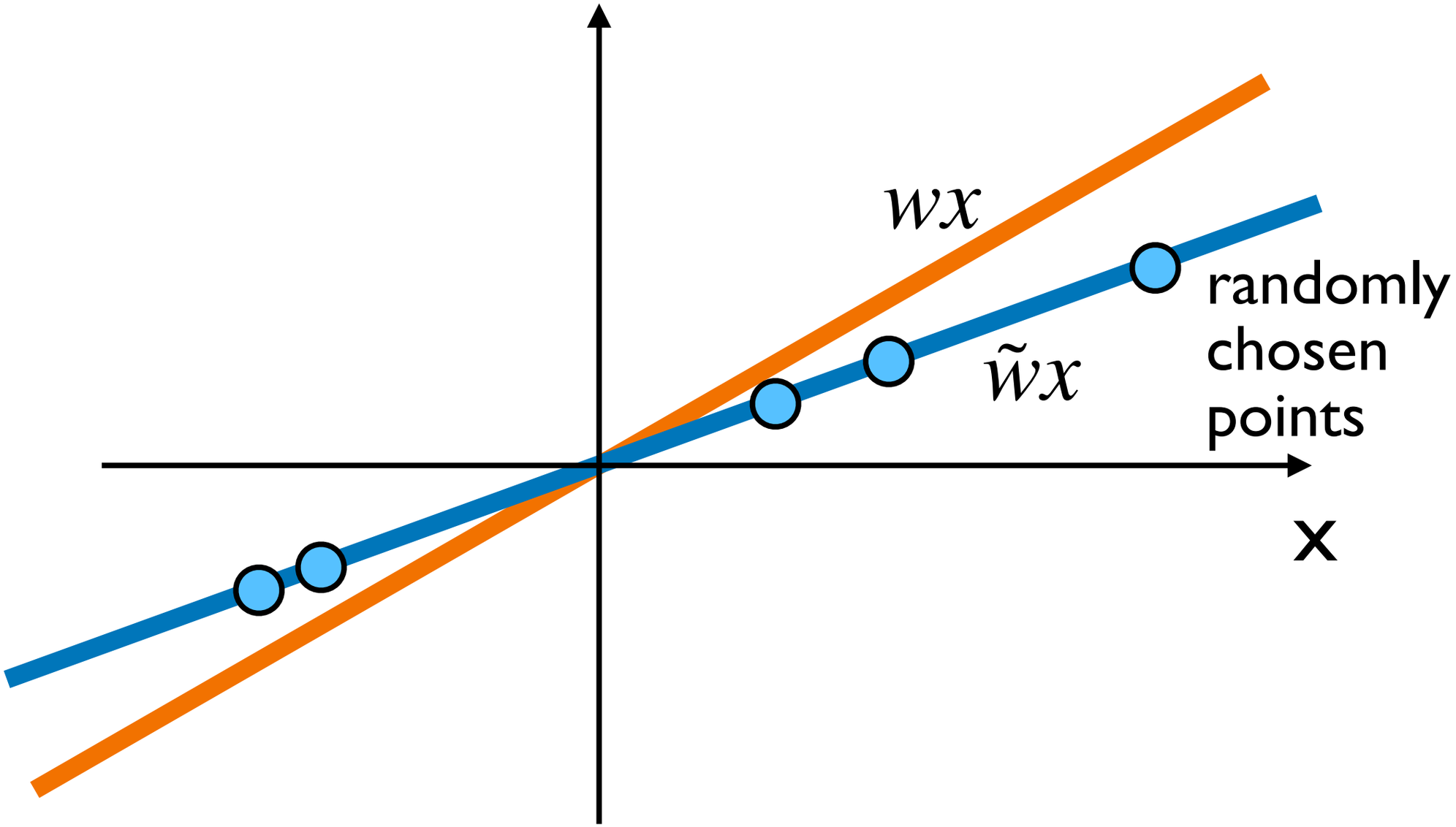




(Anirudh)




(Darshan)

# Linear fitting… ("least squares")




$$C = \frac{1}{2}\sum_{j=1}^{N}(wx_j - \tilde{w}x_j)^2$$

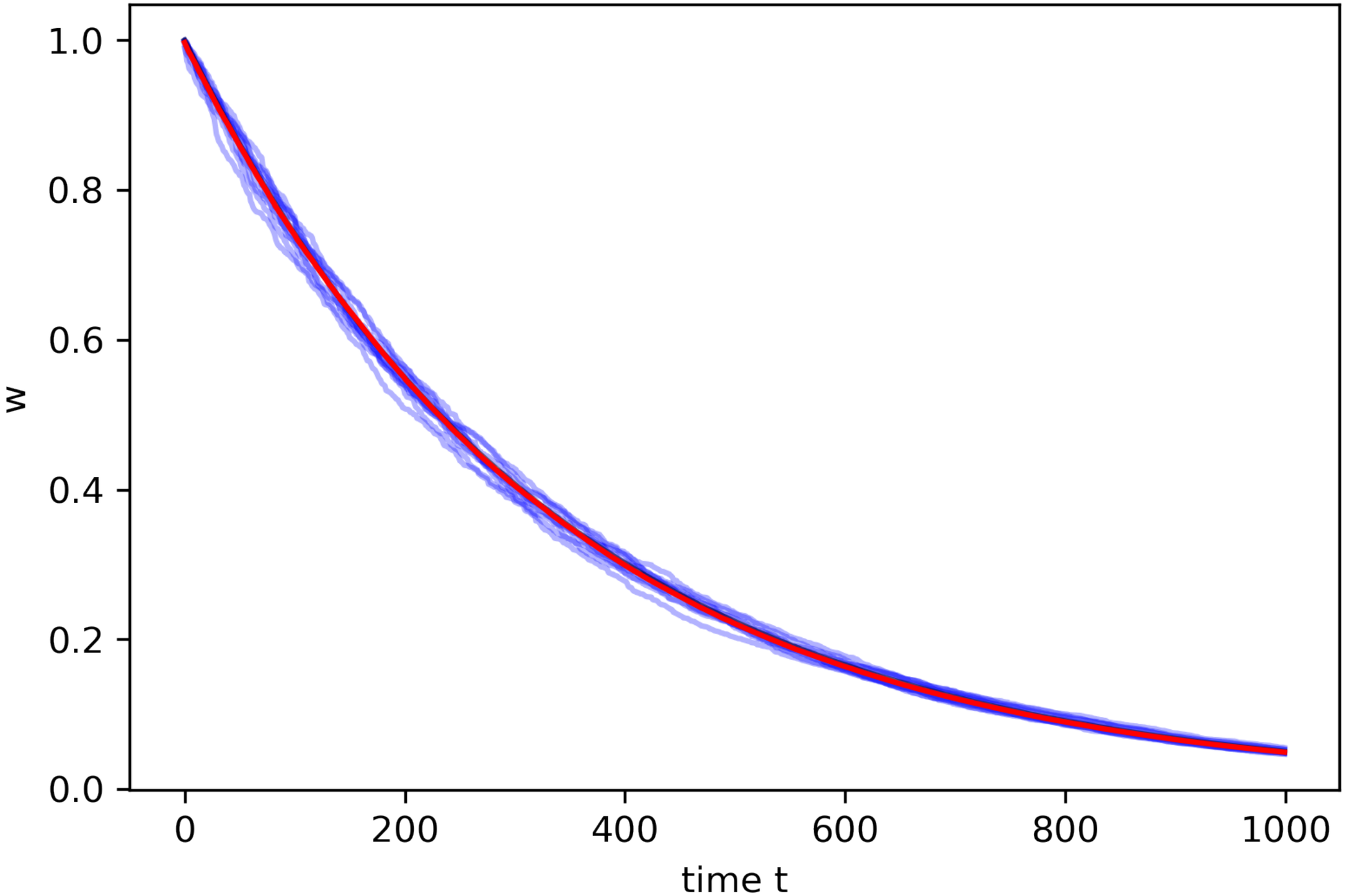
$$C = \frac{1}{2} \sum_{j=1}^{N} (w - \tilde{w})^2 x_j^2$$

$$\frac{\partial C}{\partial w} = (w - \tilde{w}) \sum_{j} x_j^2$$

$$w_{t+1} = w_t - \eta \frac{\partial C}{\partial w}$$

$$w - \tilde{w} \mapsto w$$

$$w_{t+1} = w_t (1 - \eta \sum_{j} x_j^2)$$


1.0
0.8
0.6
0.4
0.2
0.0
w
0
200
400
600
800
1000
time t

$$S = \sum_j x_j^2$$

Gaussian approximation
[central limit theorem!]

$$S = \langle S \rangle + \sqrt{\mathrm{VarS}}\,\xi$$

let simply $\left\langle x_j^2 \right\rangle = 1$

for Gaussians:
$\left\langle X^4 \right\rangle = 3 \left\langle X^2 \right\rangle^2$

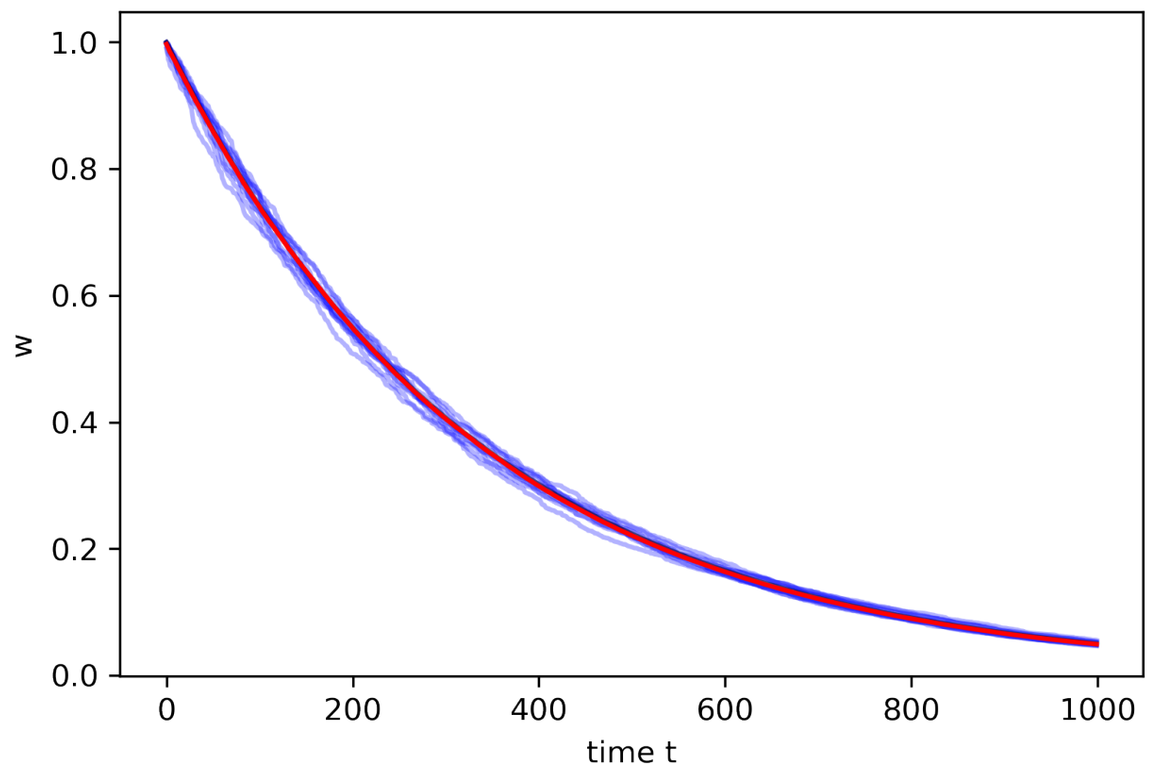
then: $S = N + \sqrt{2N}\xi$

$$w_{t+1} = w_t \left( 1 - \eta(N + \sqrt{2N}\xi_t) \right)$$

$$w_t = w_0 \exp\left(-\eta(Nt + \sqrt{2N}\sum_{\tau=0}^{t-1}\xi_\tau)\right)$$

$$\left\langle e^X \right\rangle = e^{\frac{1}{2}\langle X^2 \rangle}$$

use $\left\langle w_t \right\rangle = w_0 \exp\left(-\eta Nt + \eta^2 Nt\right)$



# Lecture 3 Homework

(will be briefly discussed in the online session for lecture 4)

Machine Learning for Physicists

* 1 Train on some function or image and explore the role of various neurons (by switching weights on/off like in the lecture); use the keras training visualization notebook

* 2 Produce a simple network with keras that can distinguish a 1D plot of a random Gaussian from a 1D plot of a random Lorentzian! (or other classes of functions)

3 Produce a simple network with keras that can distinguish a 2D picture of a randomly placed square from that of a randomly placed circle! What happens if you add noise to the images?

* minimum