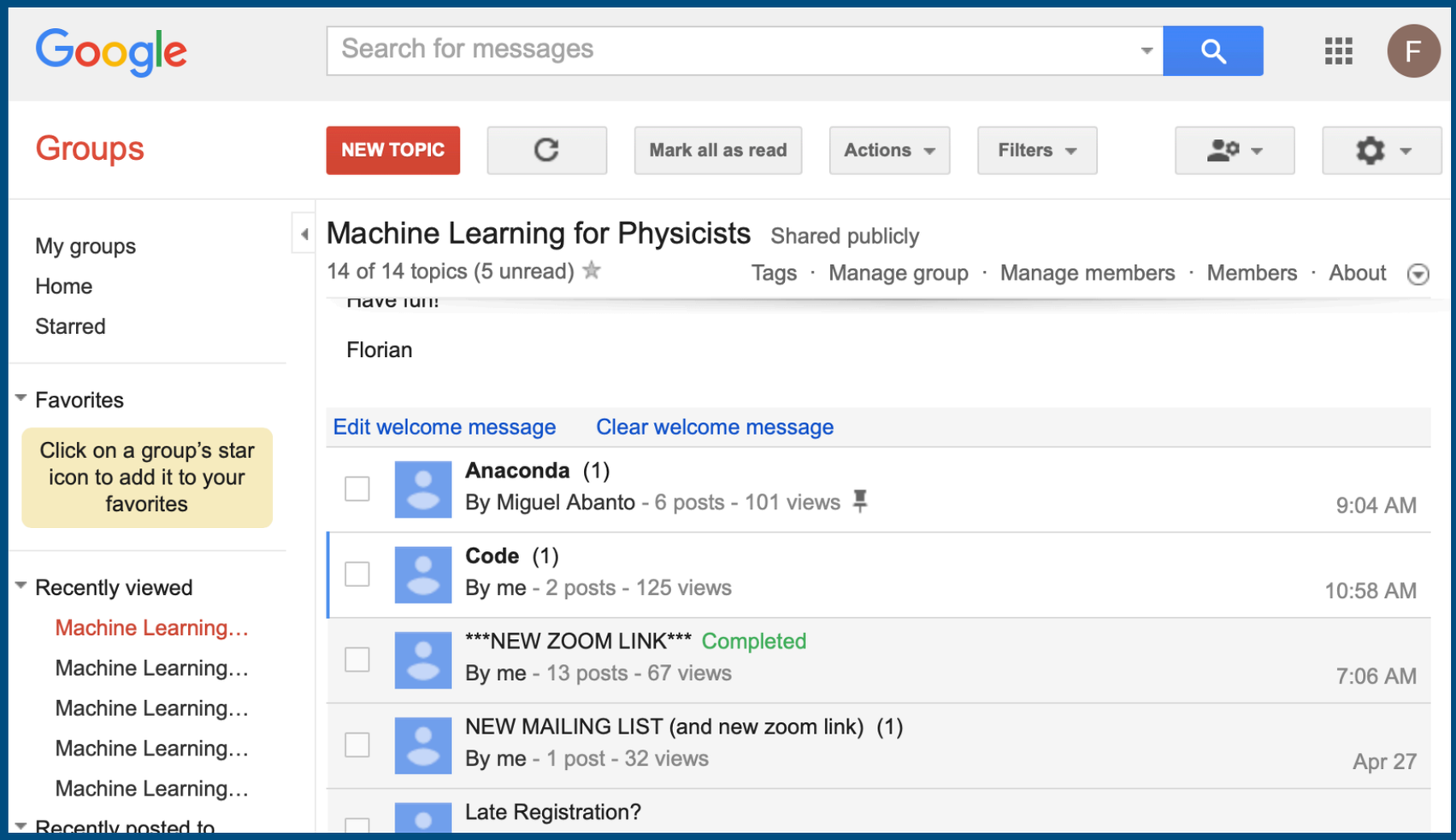
Machine Learning for Physicists

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

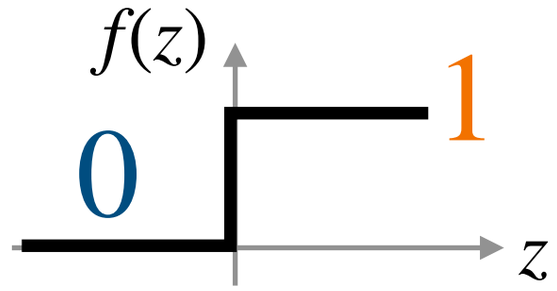


Machine Learning for Physicists

Quiz: single-layer neural networks

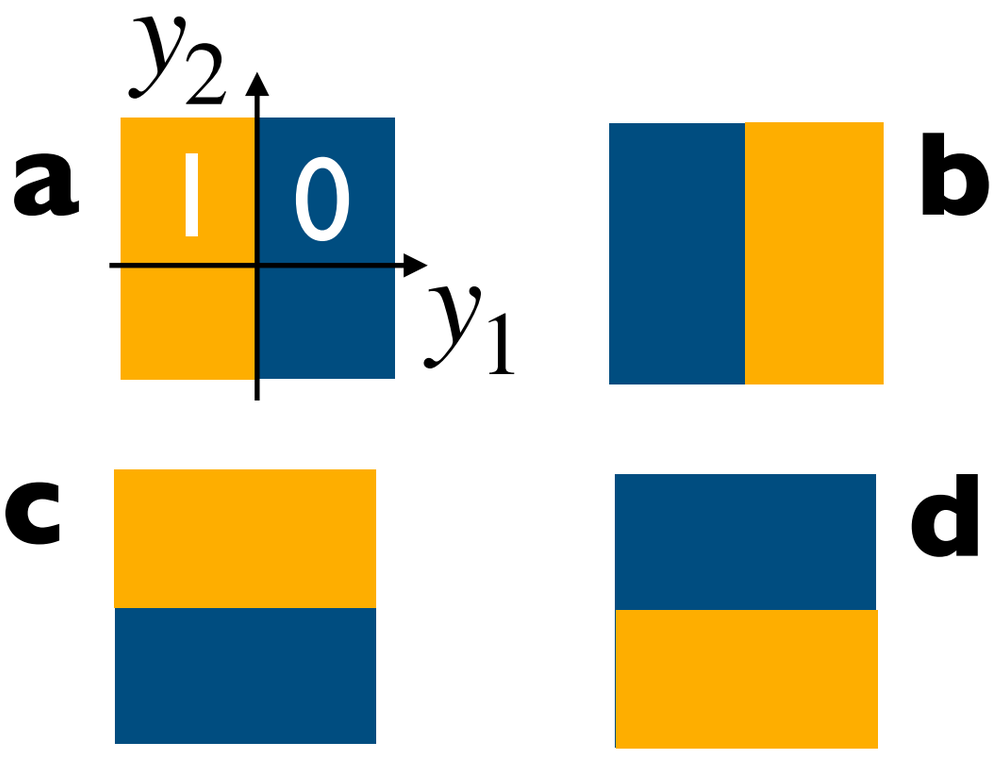
Machine Learning for Physicists

$f(z)$ is the step function:



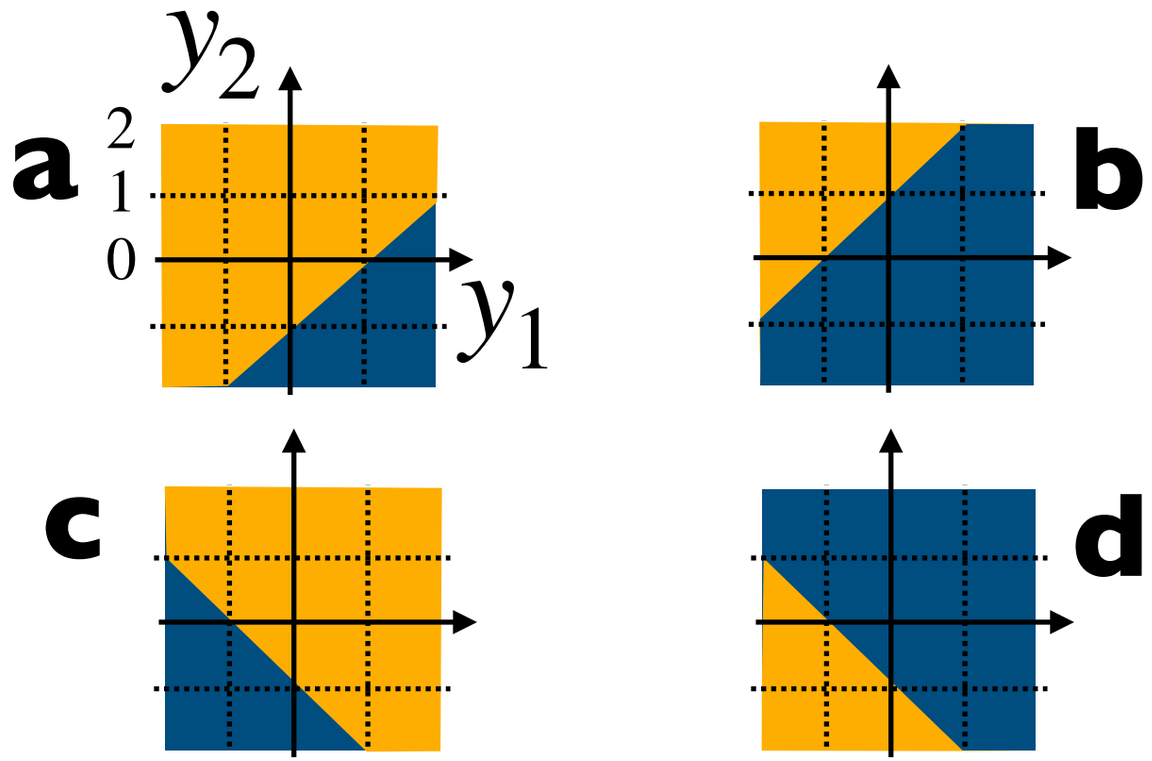
1

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

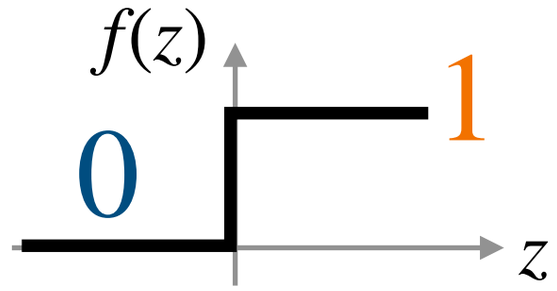


2

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

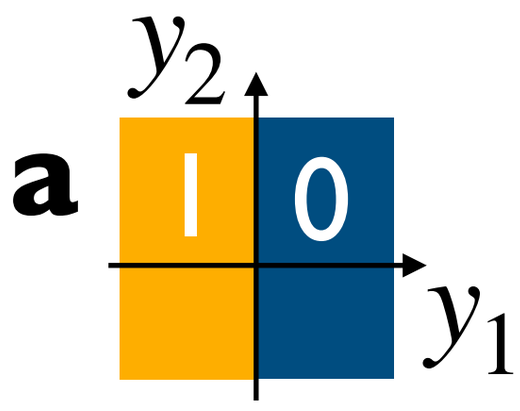


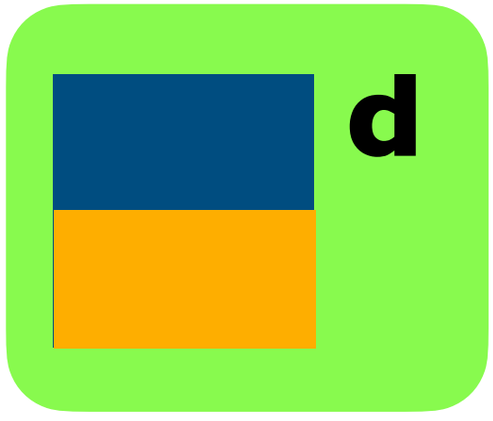
$f(z)$ is the step function:



1

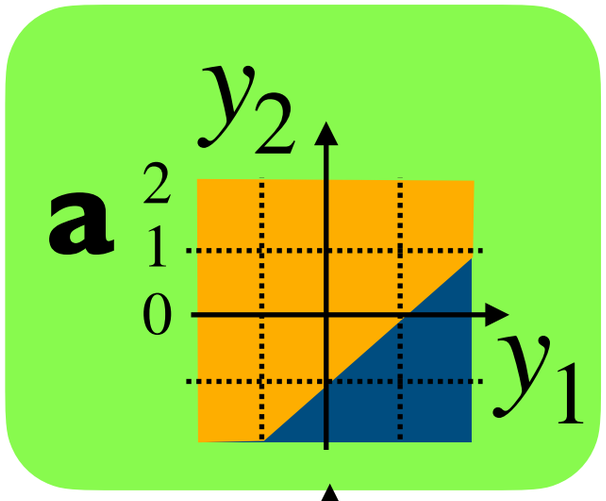
Which of these outputs is generated by $y^{\text{out}} = f(-y_2)$?

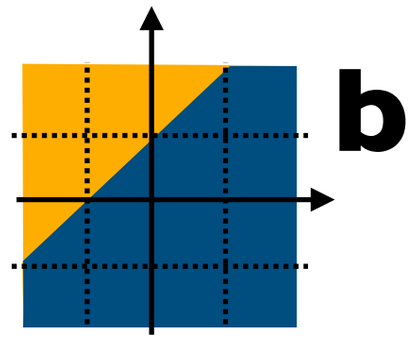


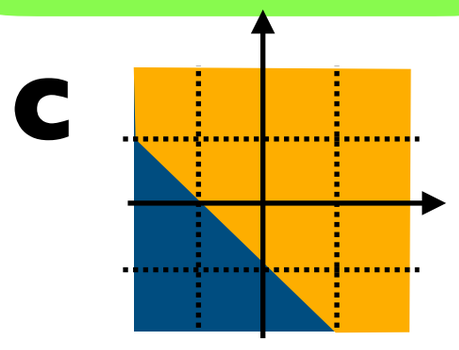


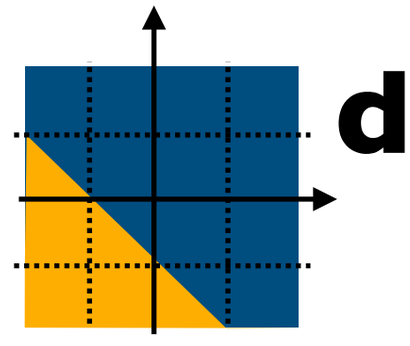
2

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

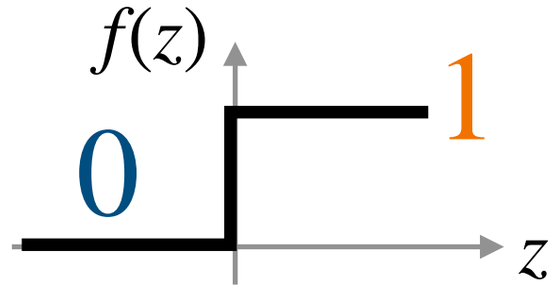




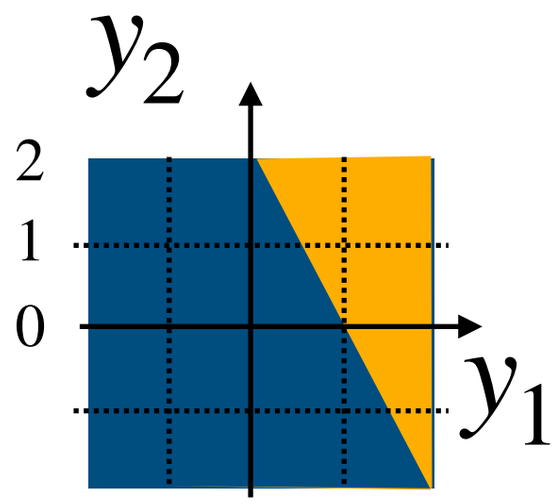




$f(z)$ is the step function:



1 How can we obtain this output?



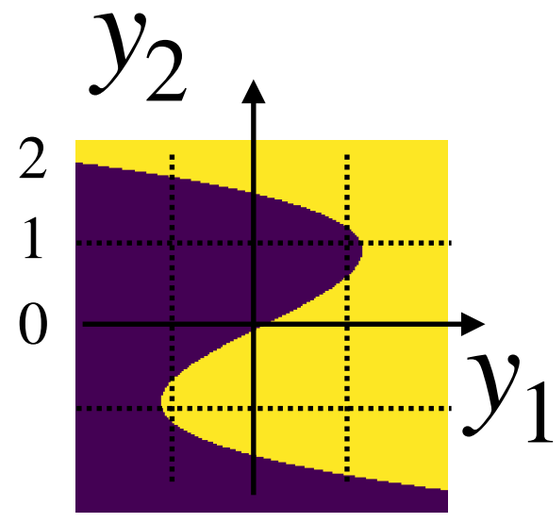
a $y^{\text{out}} = f(y_1 + y_2 - 2)$

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?



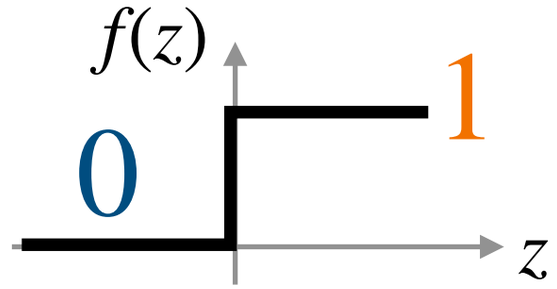
a $y^{\text{out}} = f(-2y_2 + y_1)$

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

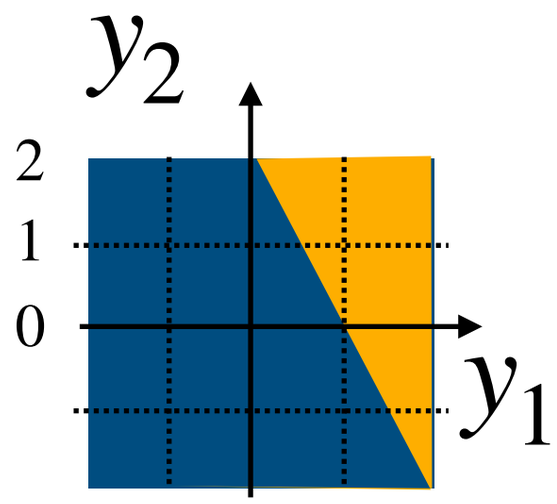
c $y^{\text{out}} = f(y_2^3 - 2y_2 + y_1)$

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?



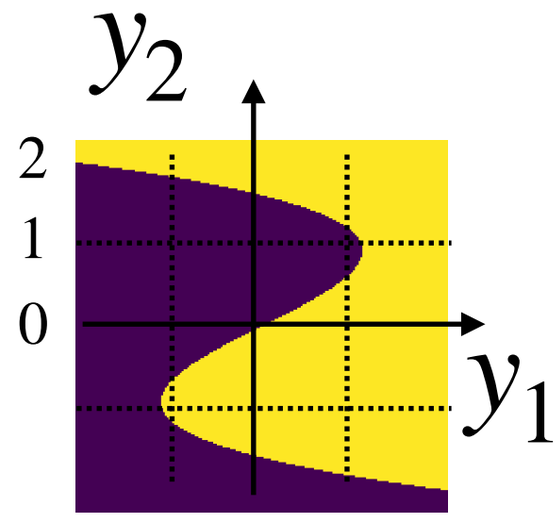
a $y^{\text{out}} = f(y_1 + y_2 - 2)$

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=?
```

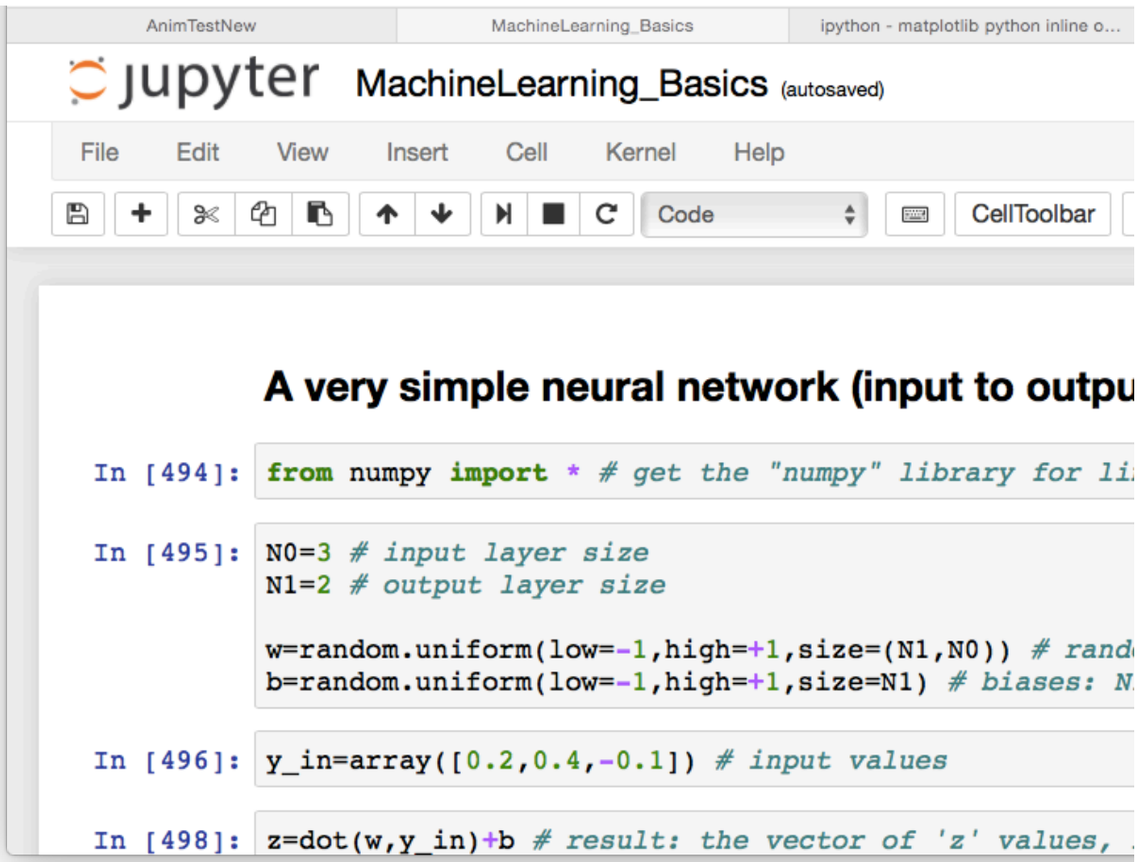
a `[[11,22],[19,50]]`

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

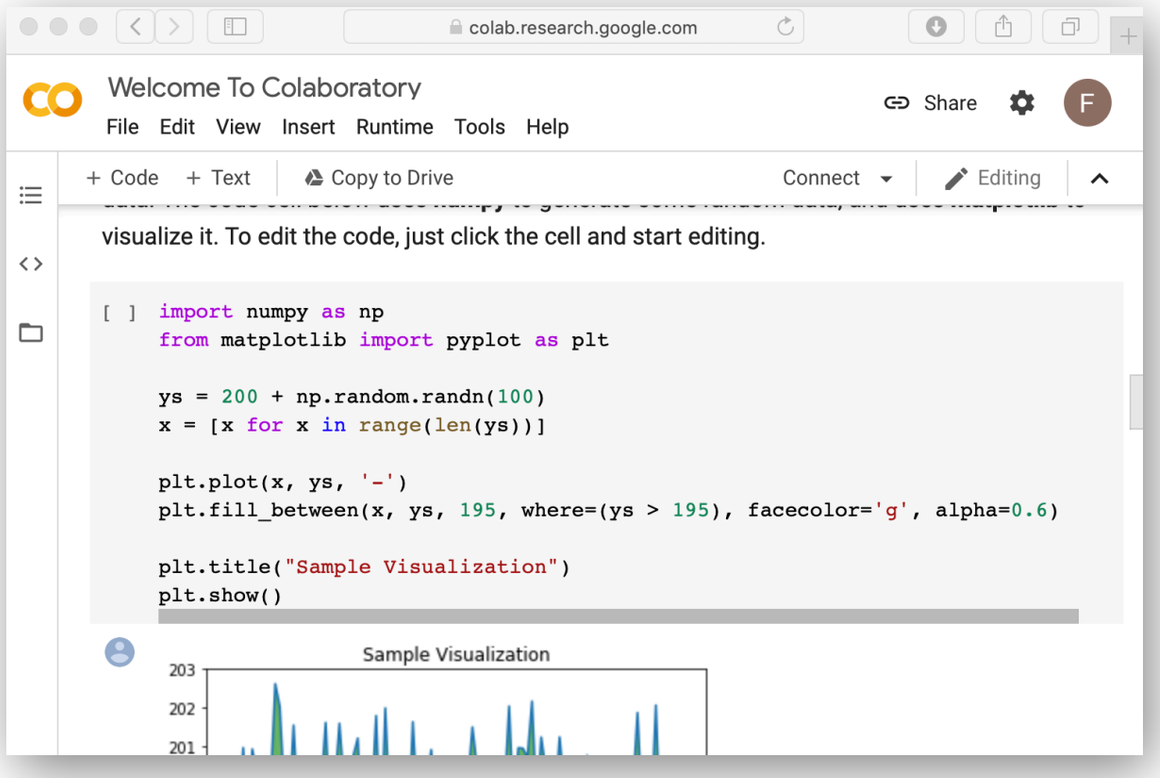
c `[[19,22],[50,44]]`

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).

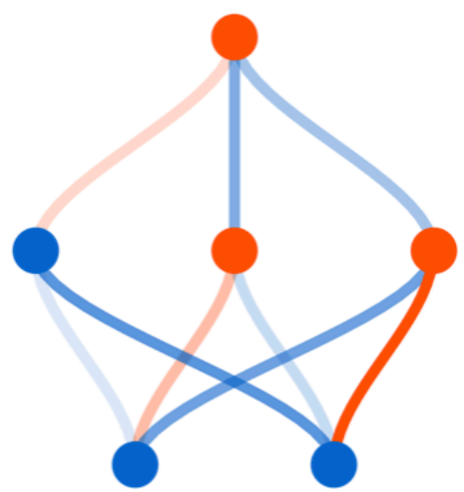
Notebooks for tutorials:
Tutorial: Network Visualization
Tutorial: Curve Fitting

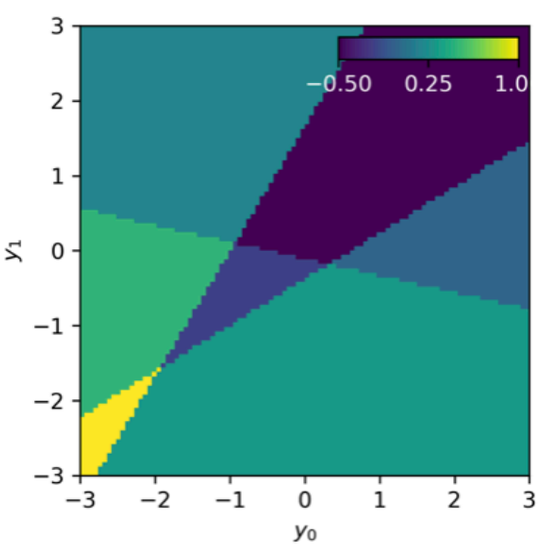
Settings Help

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])
```





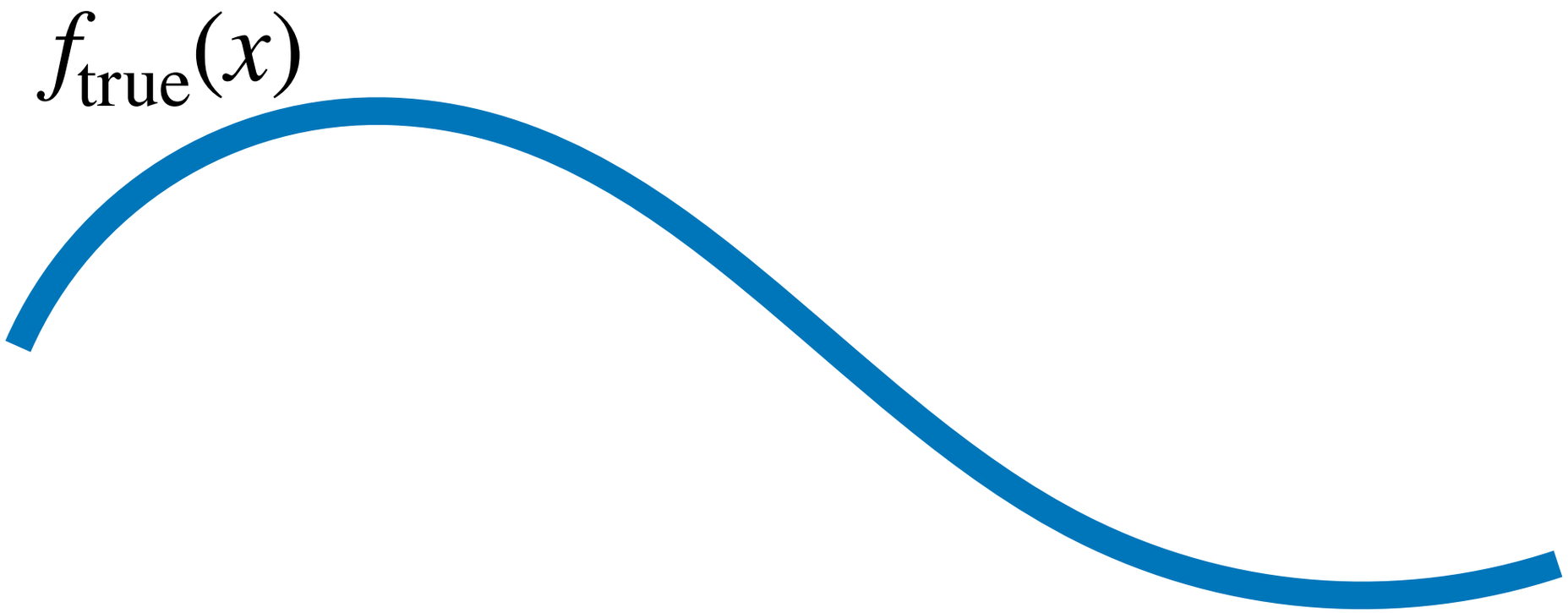
0 $_ 7 Python Mode: Ln 1, 01_tutorial_NetworkVisua

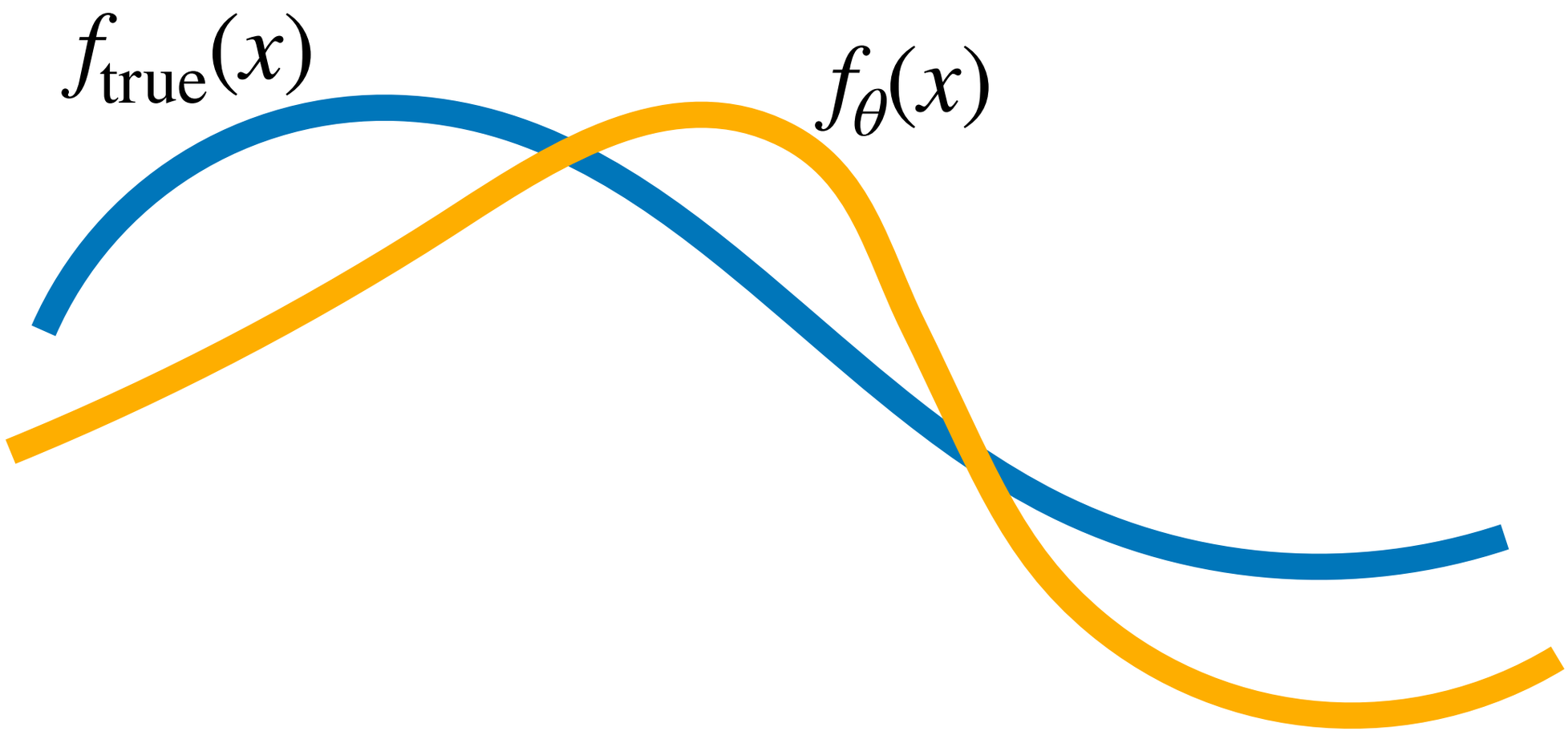
(1) Try to construct a square!

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

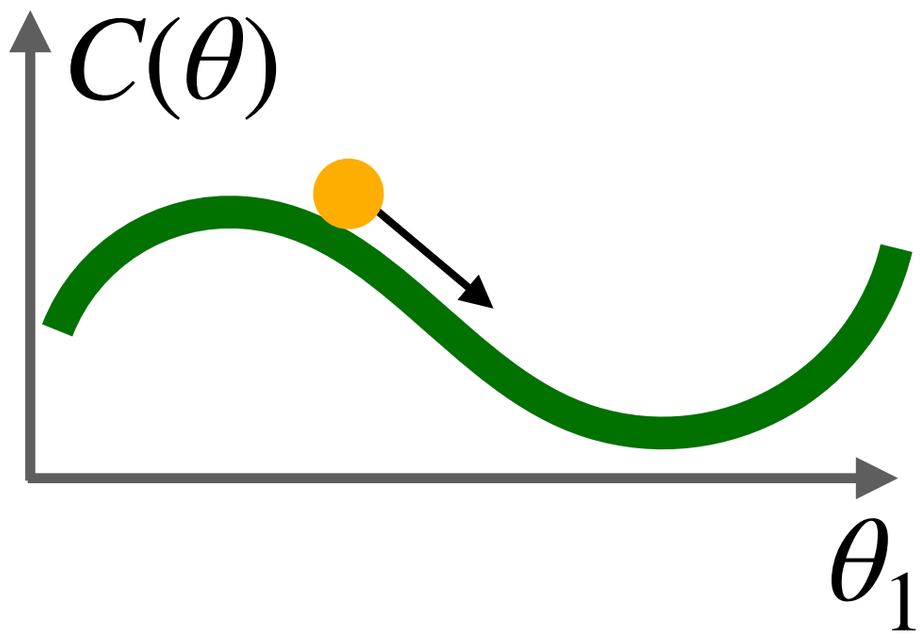
"least squares fitting":
consider quadratic deviation

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

(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

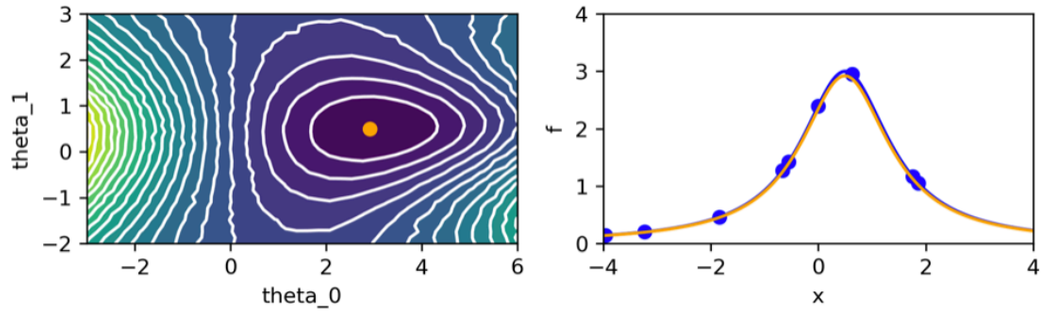
Tutorial: Curve Fitting

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)
```



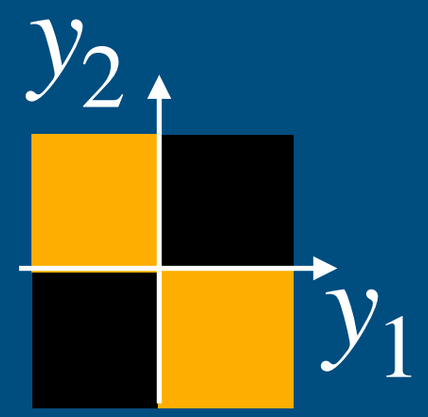
Your own examples!

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(!)

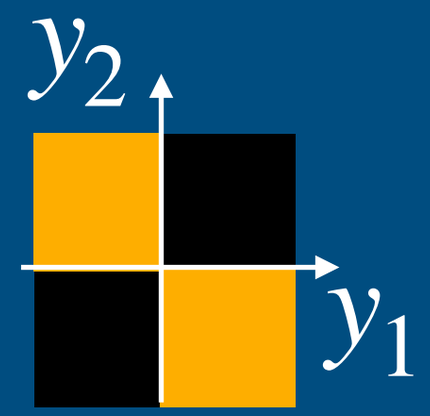
* minimum

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)
```

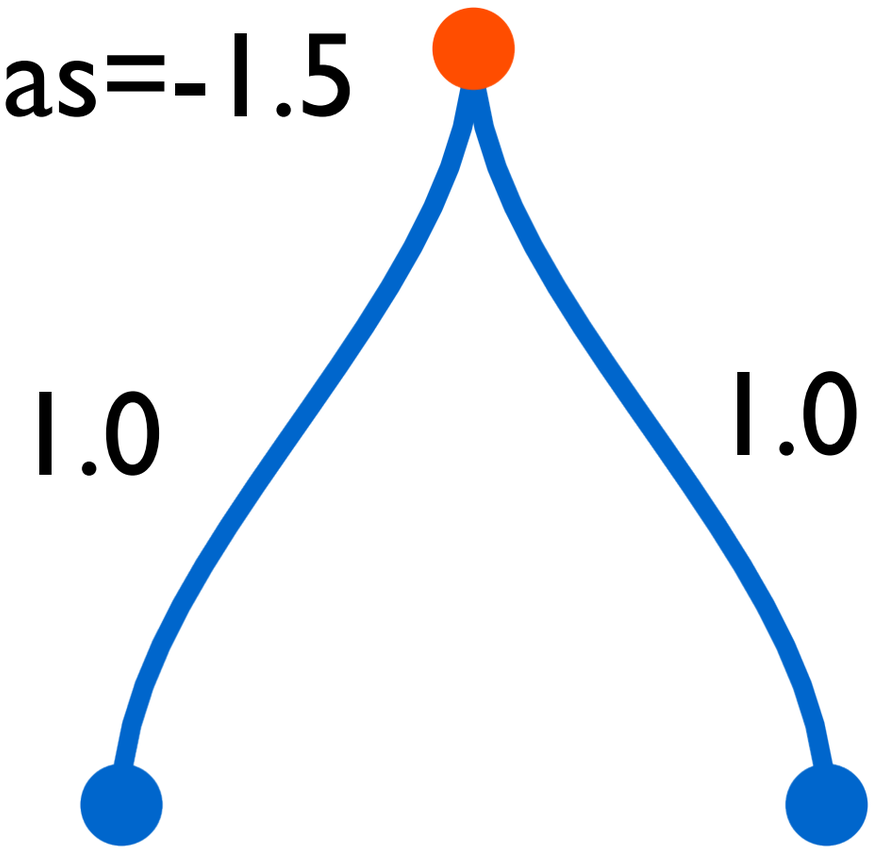
f = step function

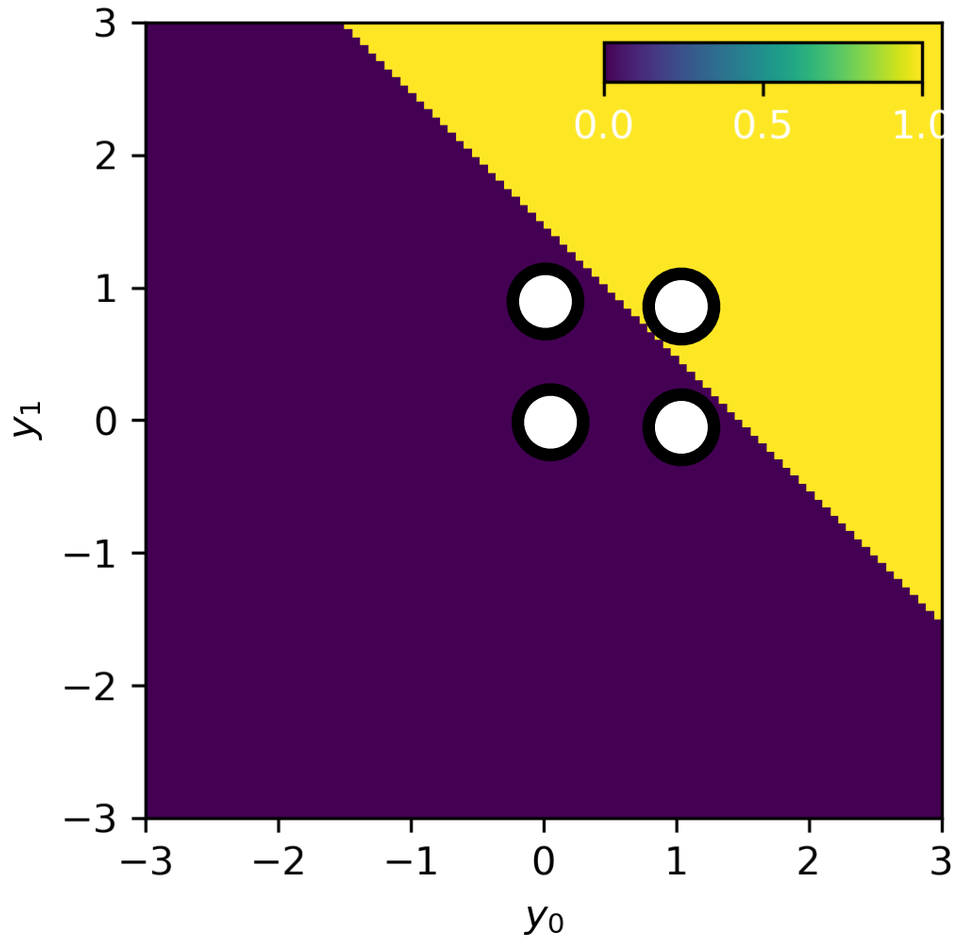
"warm up" AND function

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

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

bias=-1.5

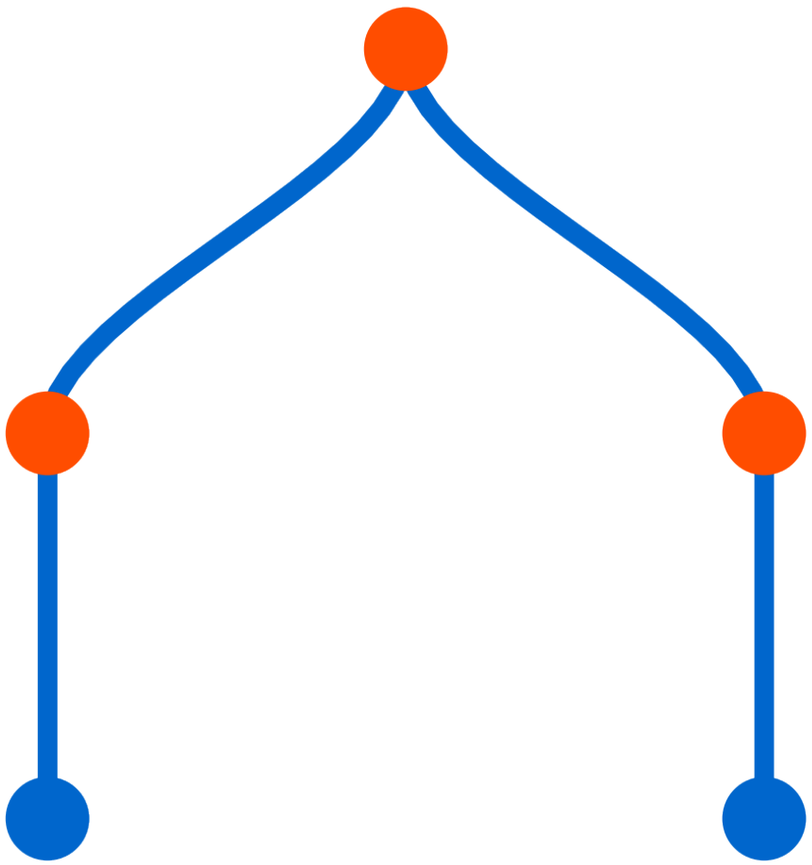


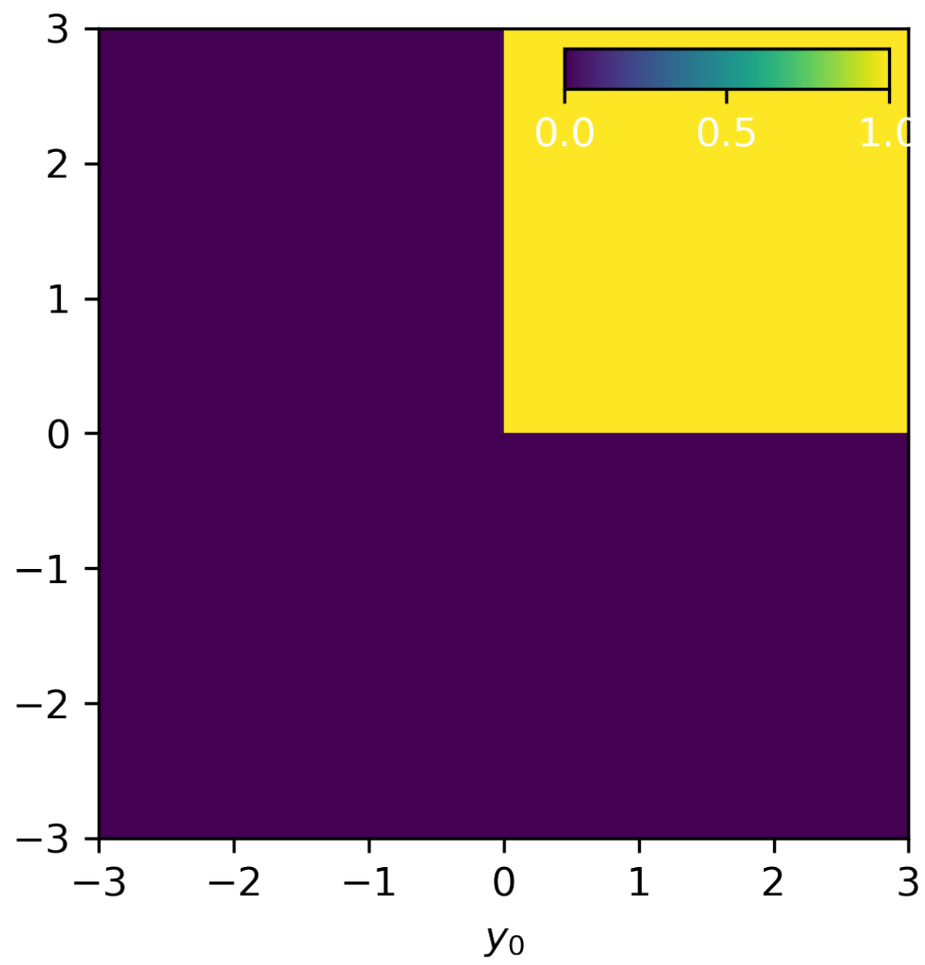



```
weights=[ [    [1.0,1.0]     ] ],

biases=[ [-1.5] ]
```

"warm up" alternative AND function

+1 exactly for y0>0 AND y1>0



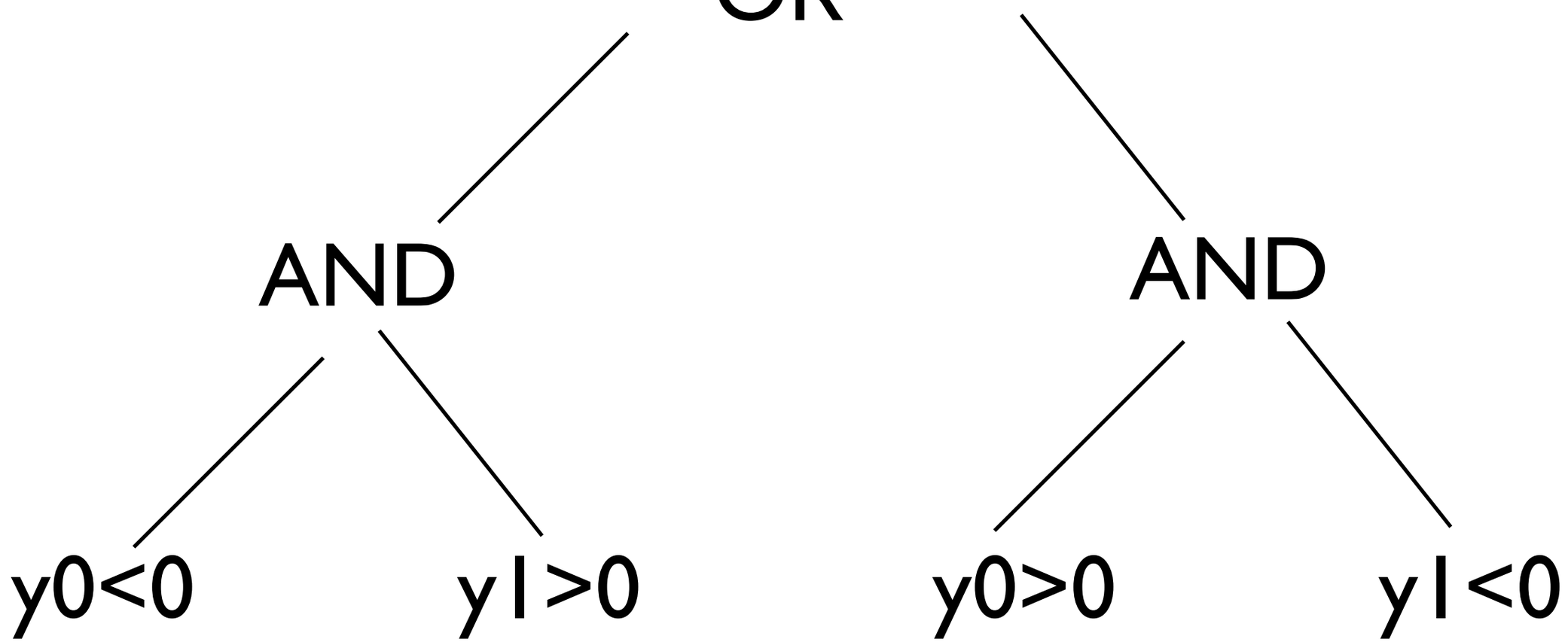



```
weights=[ [ [1.0,0.0], [0.0,1.0] ],
[ [1.0,1.0] ] ],
biases=[  [0.0,0.0], [-1.5] ]
```

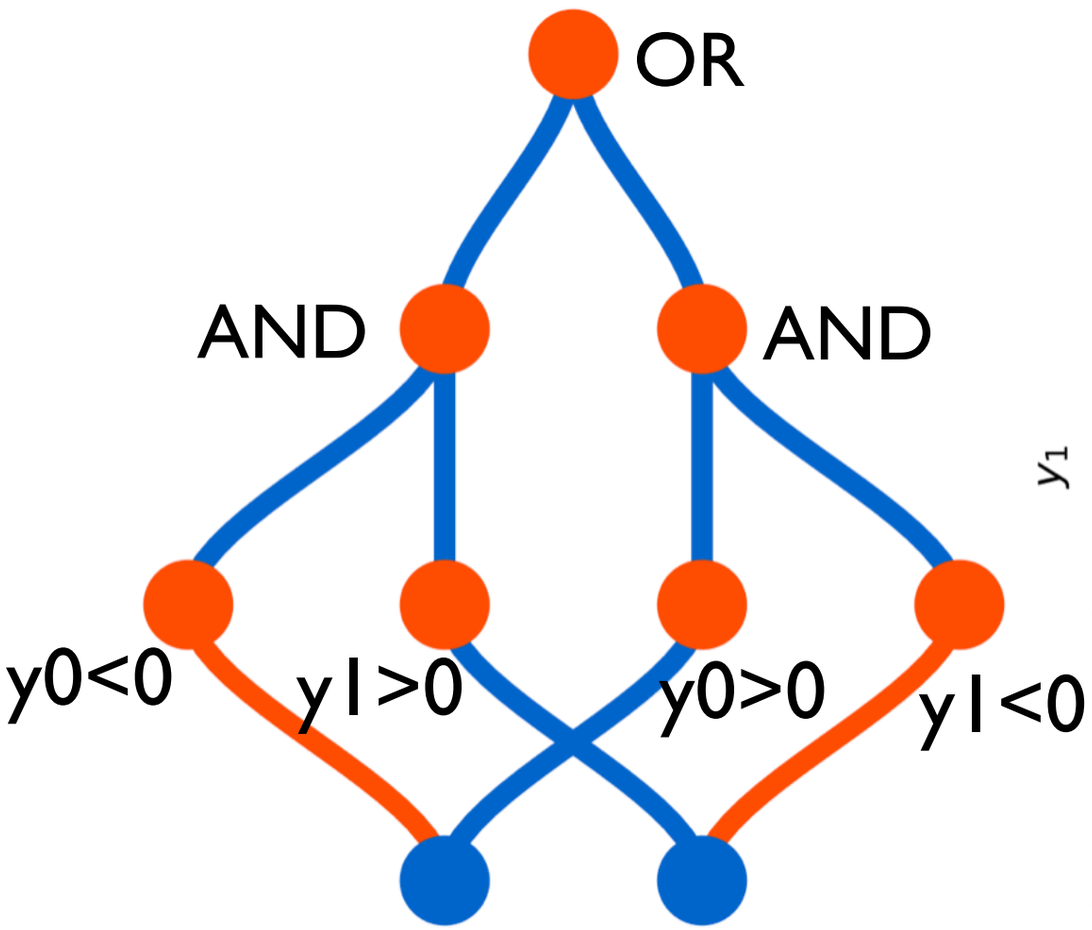
the XOR function

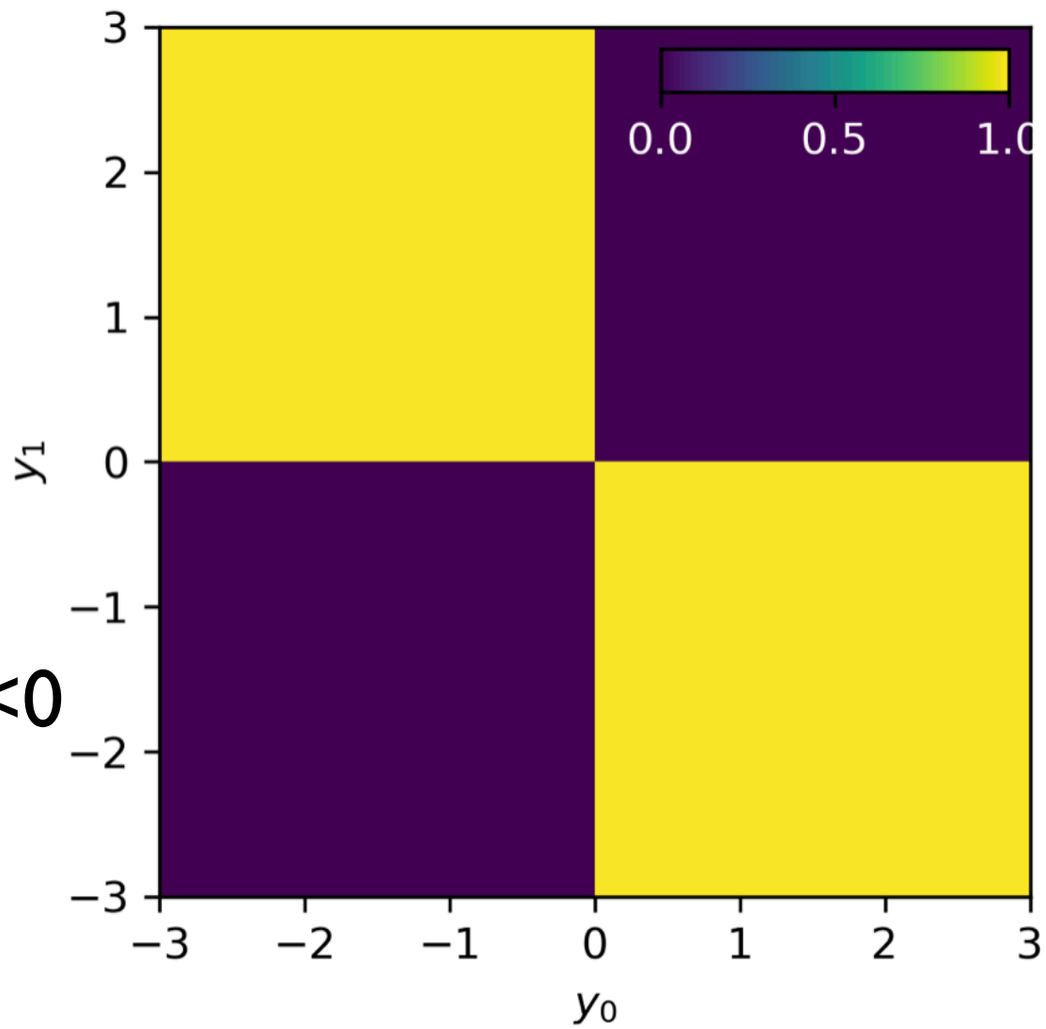
"exclusive or": +1 for y0*y1<0

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

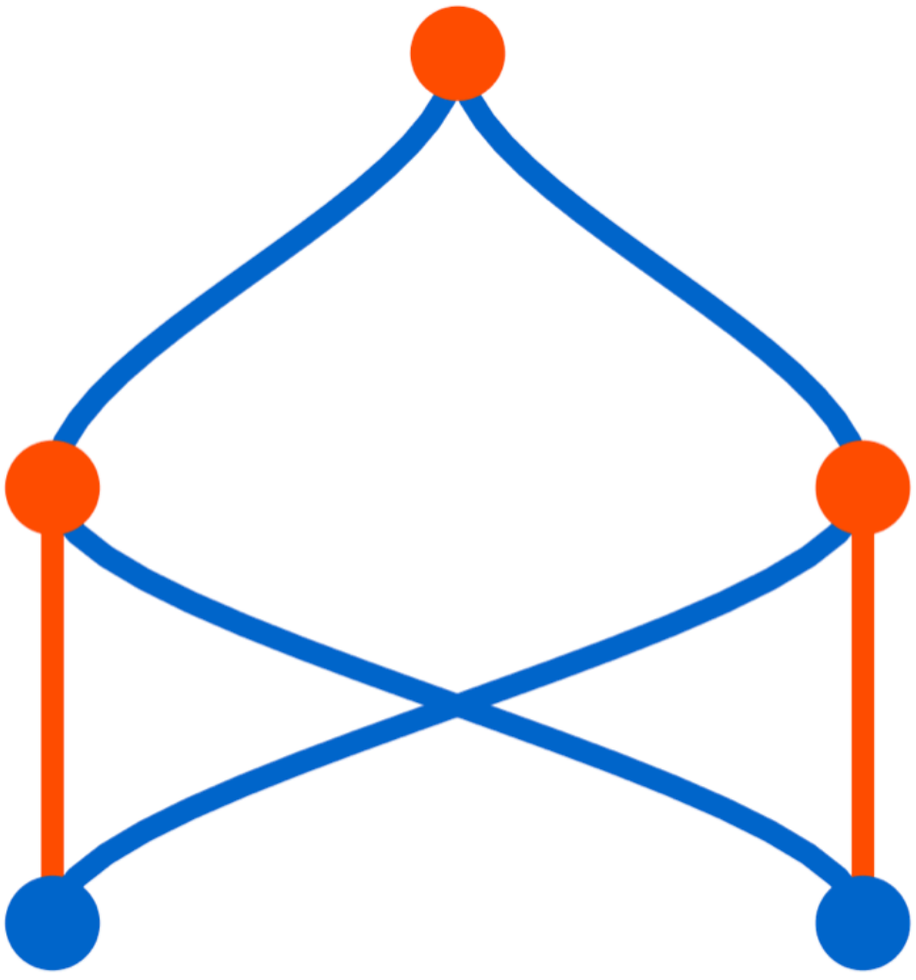


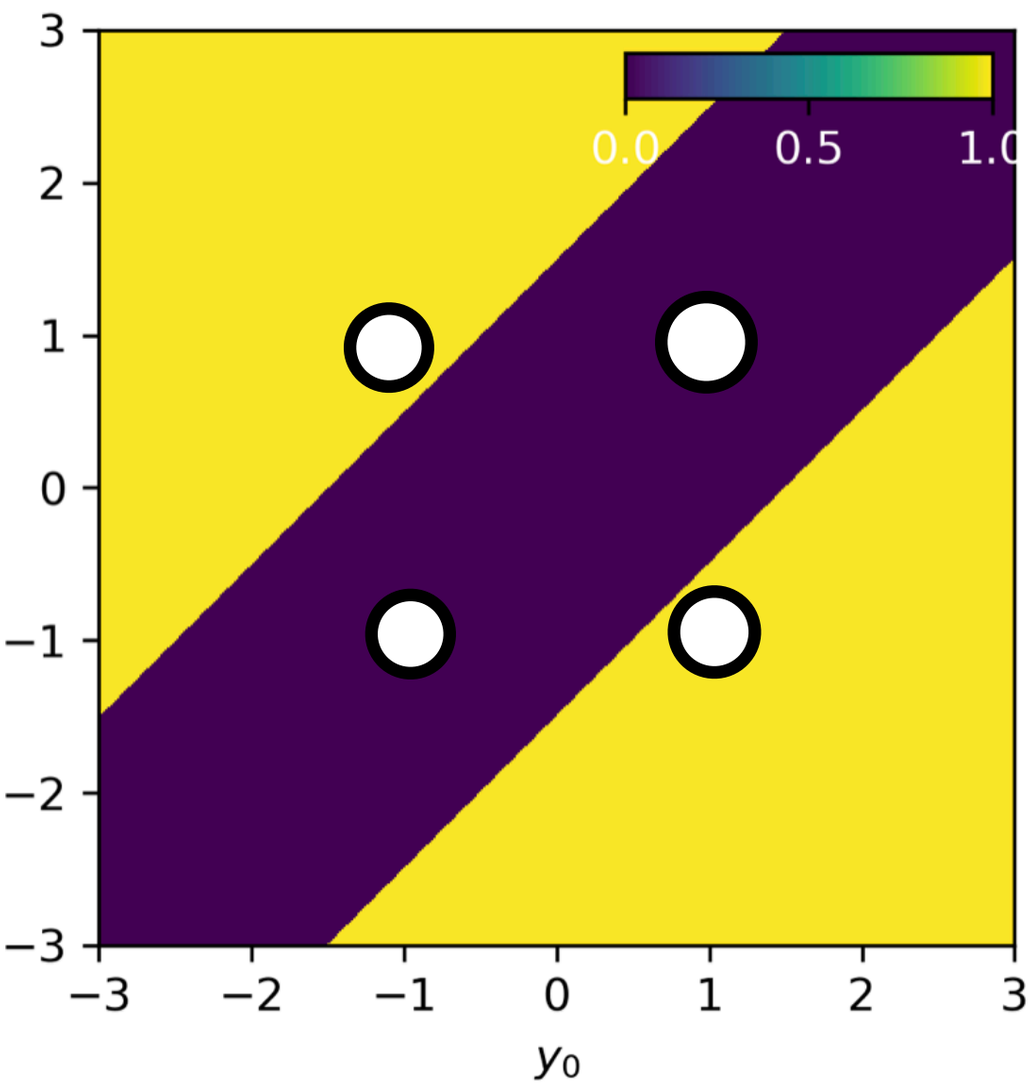
```
# XOR = (y0<0 AND y1>0) OR (y0>0 AND y1<0)
# 4 neurons in first hidden layer
# 2 neurons in second hidden layer

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)
```

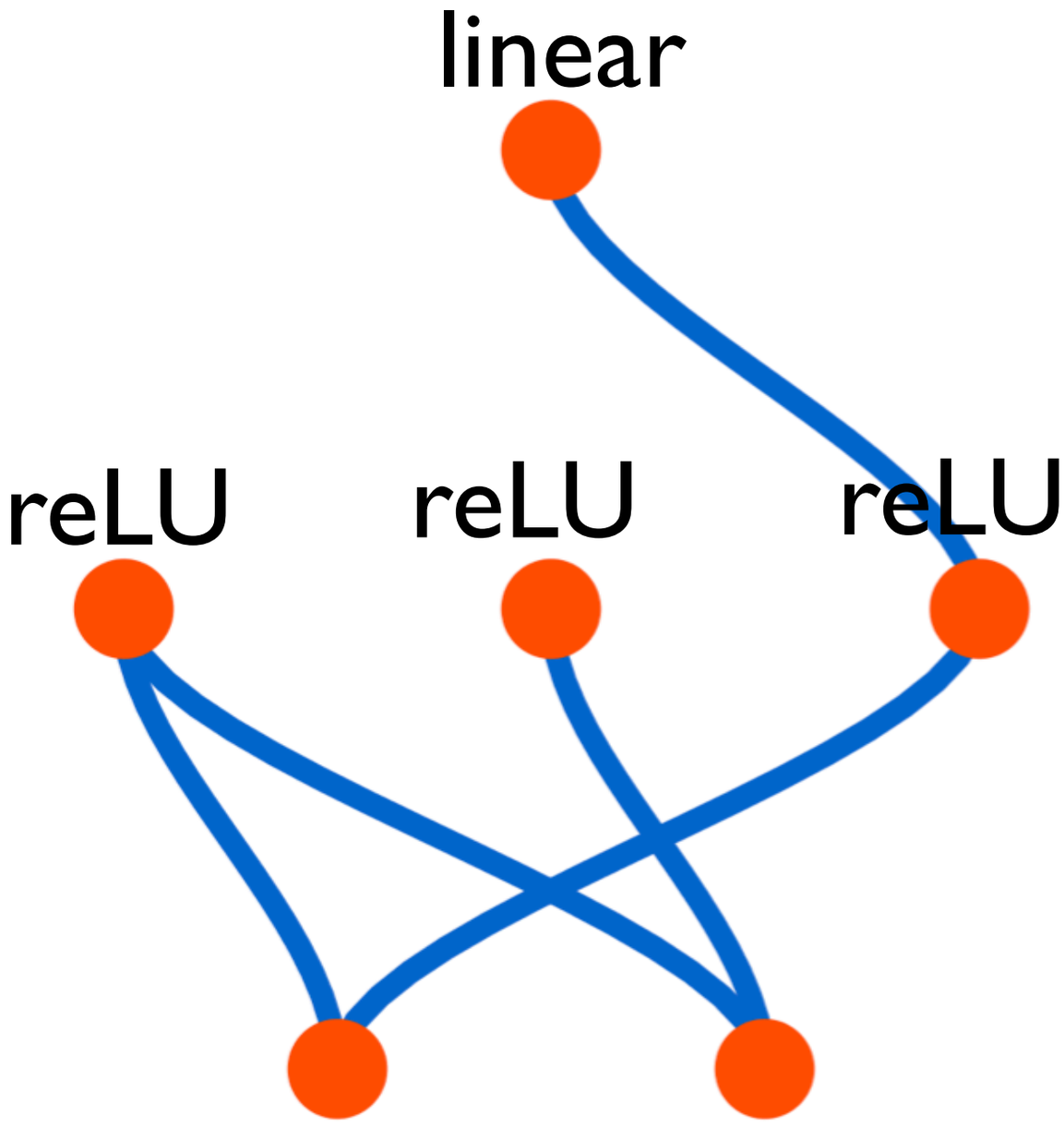


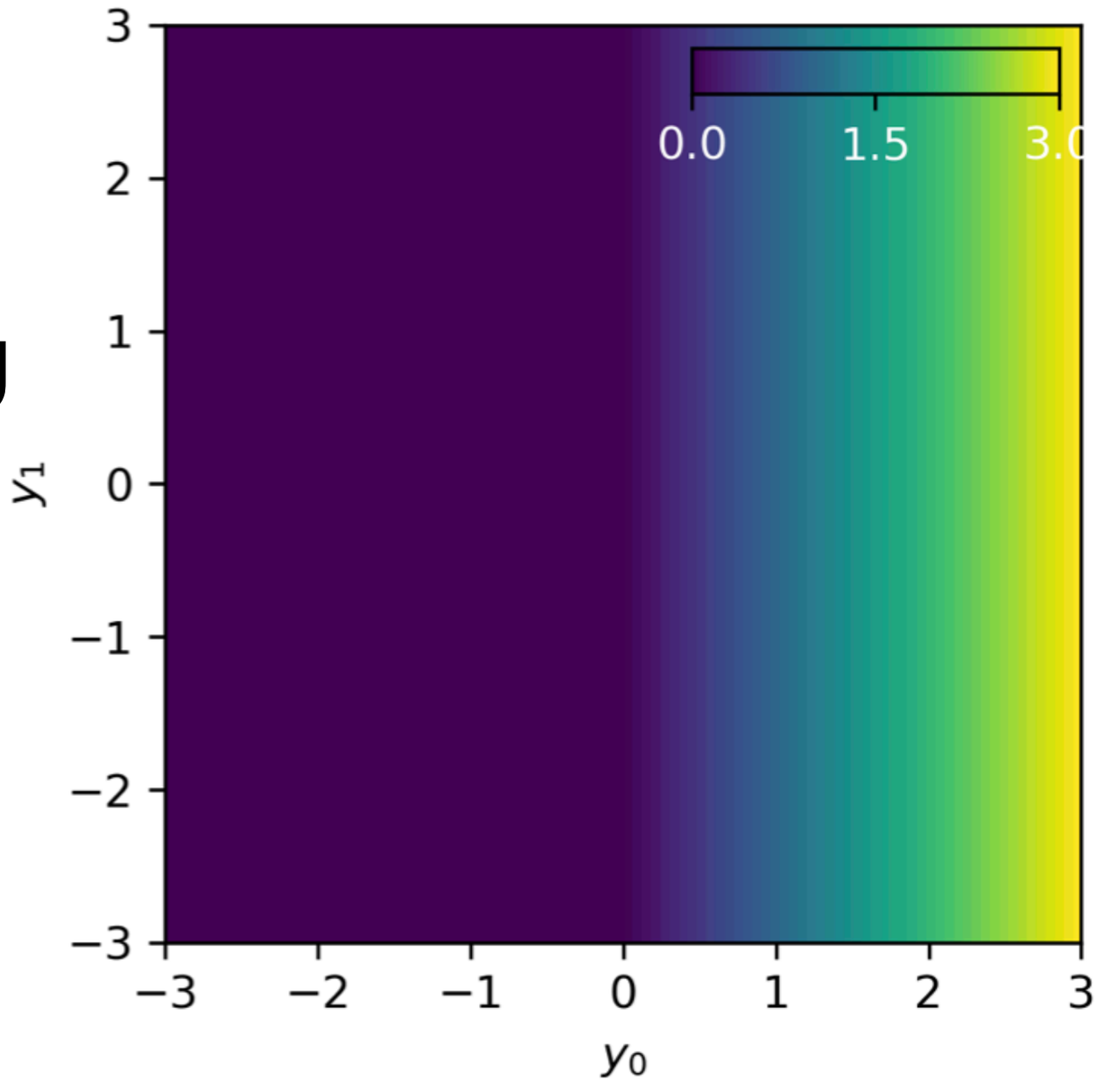


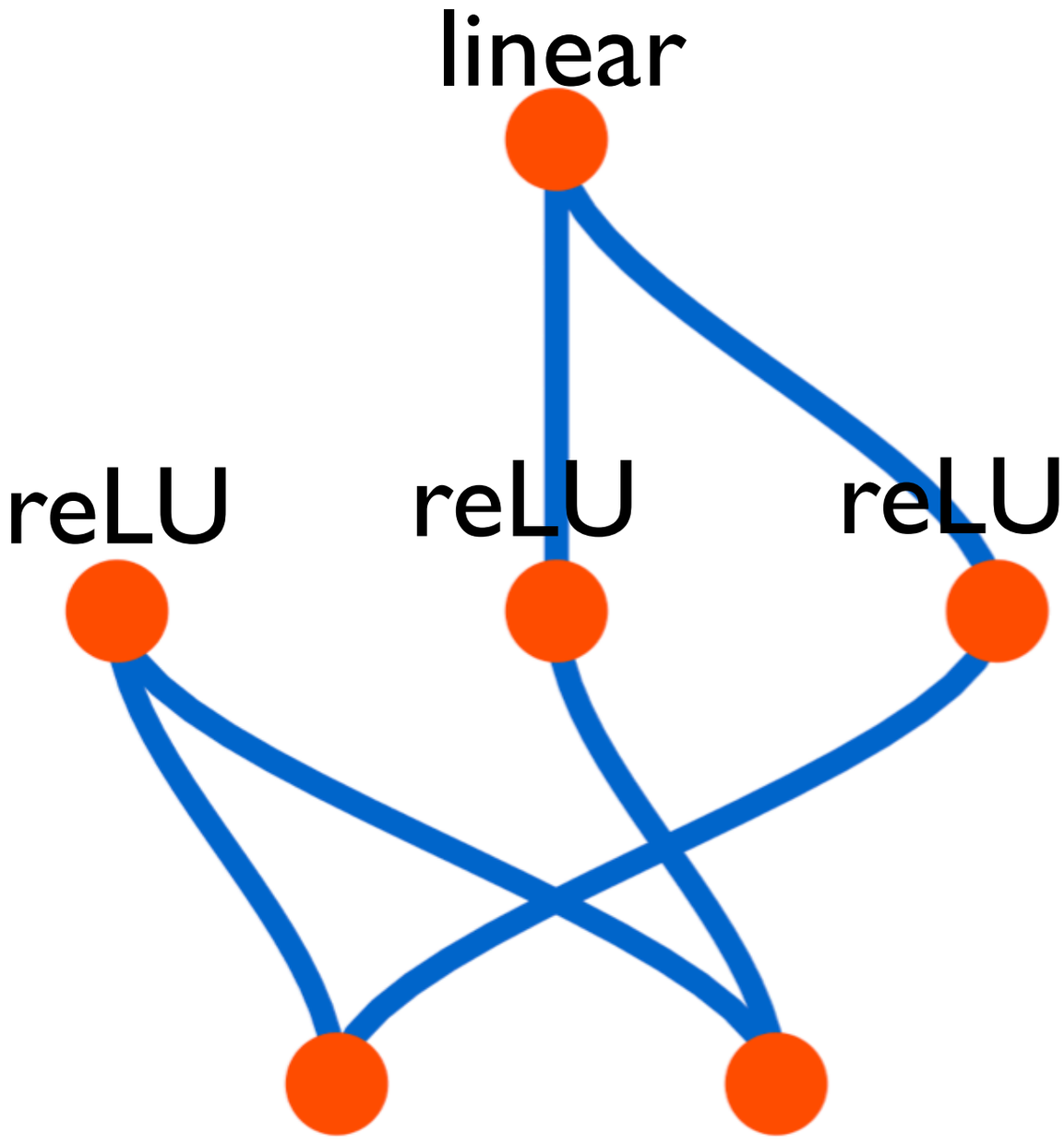
```
# Using the alternative AND function, where the right
# result comes about only for the specific inputs (-1,-1), (-1,+1)
# etc etc

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)
```

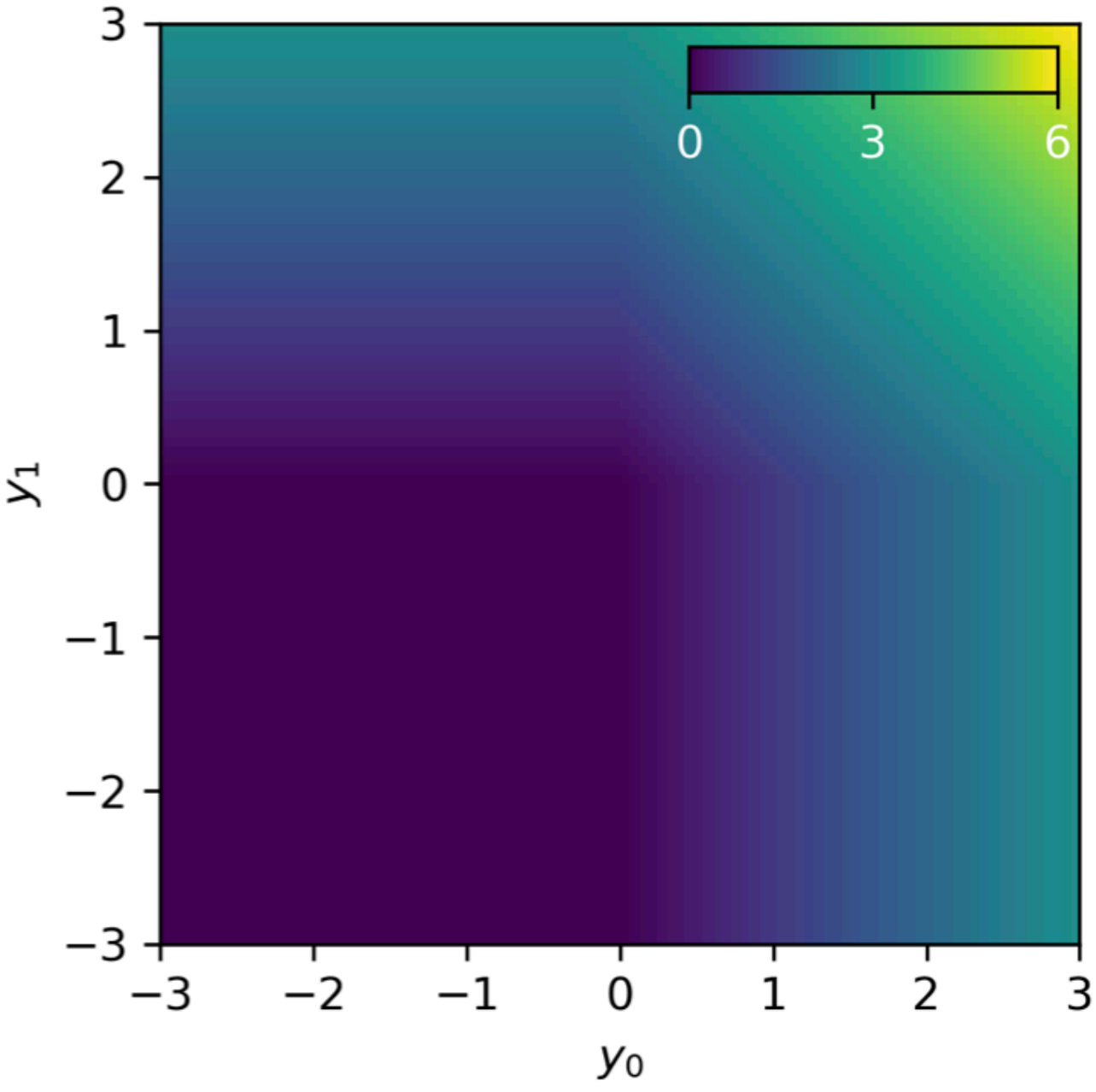


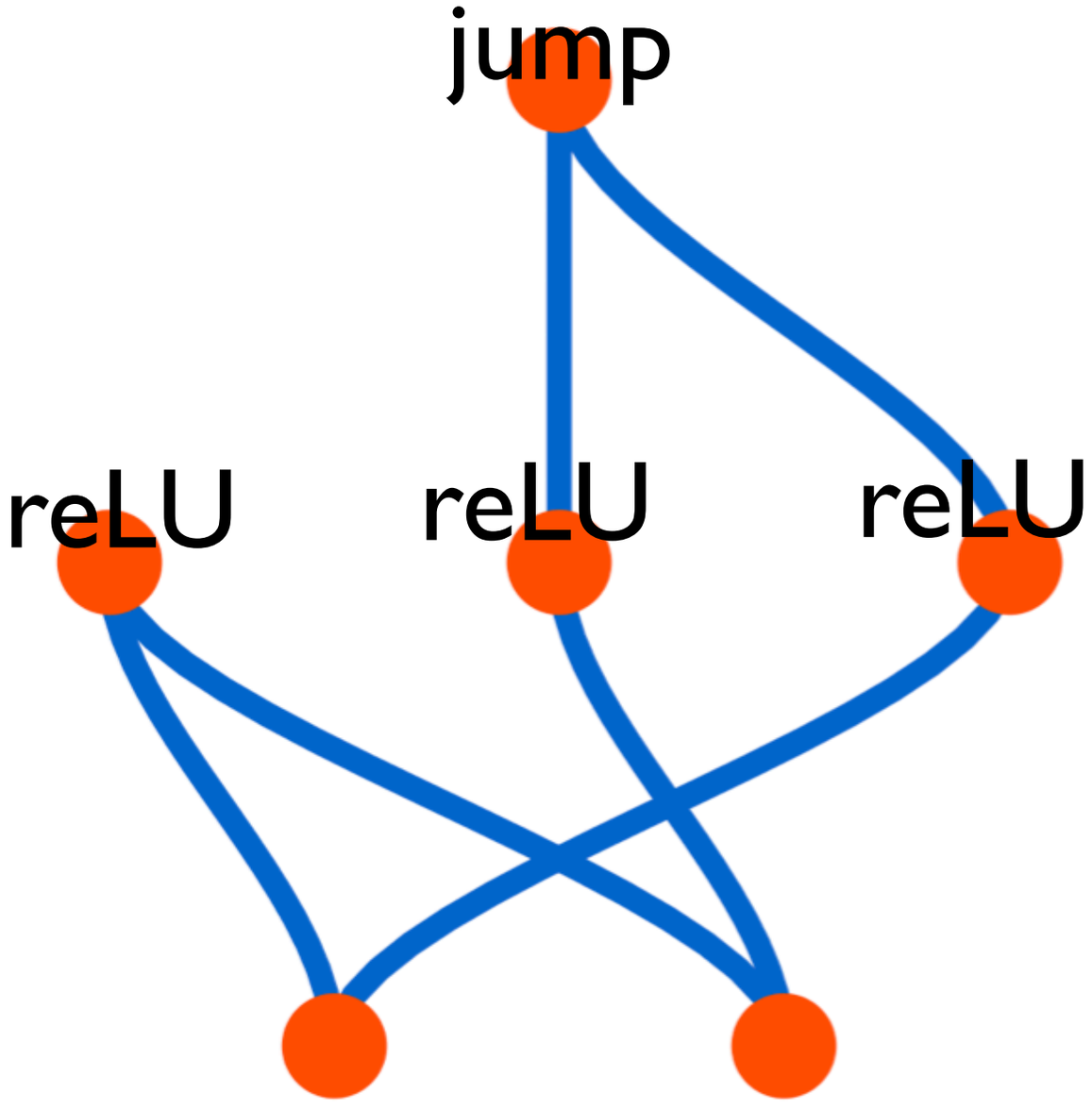


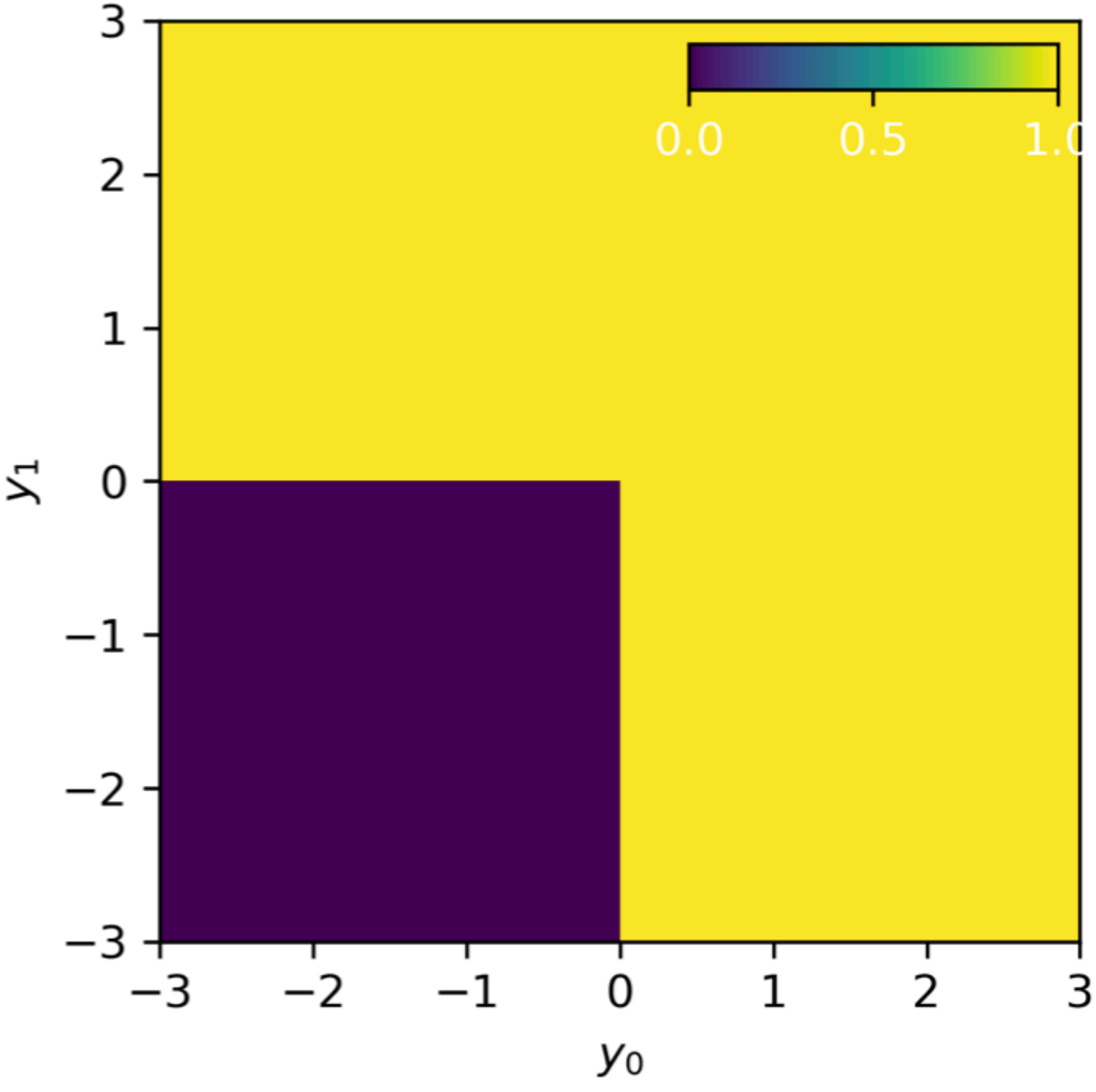

linear
reLU
reLU
reLU

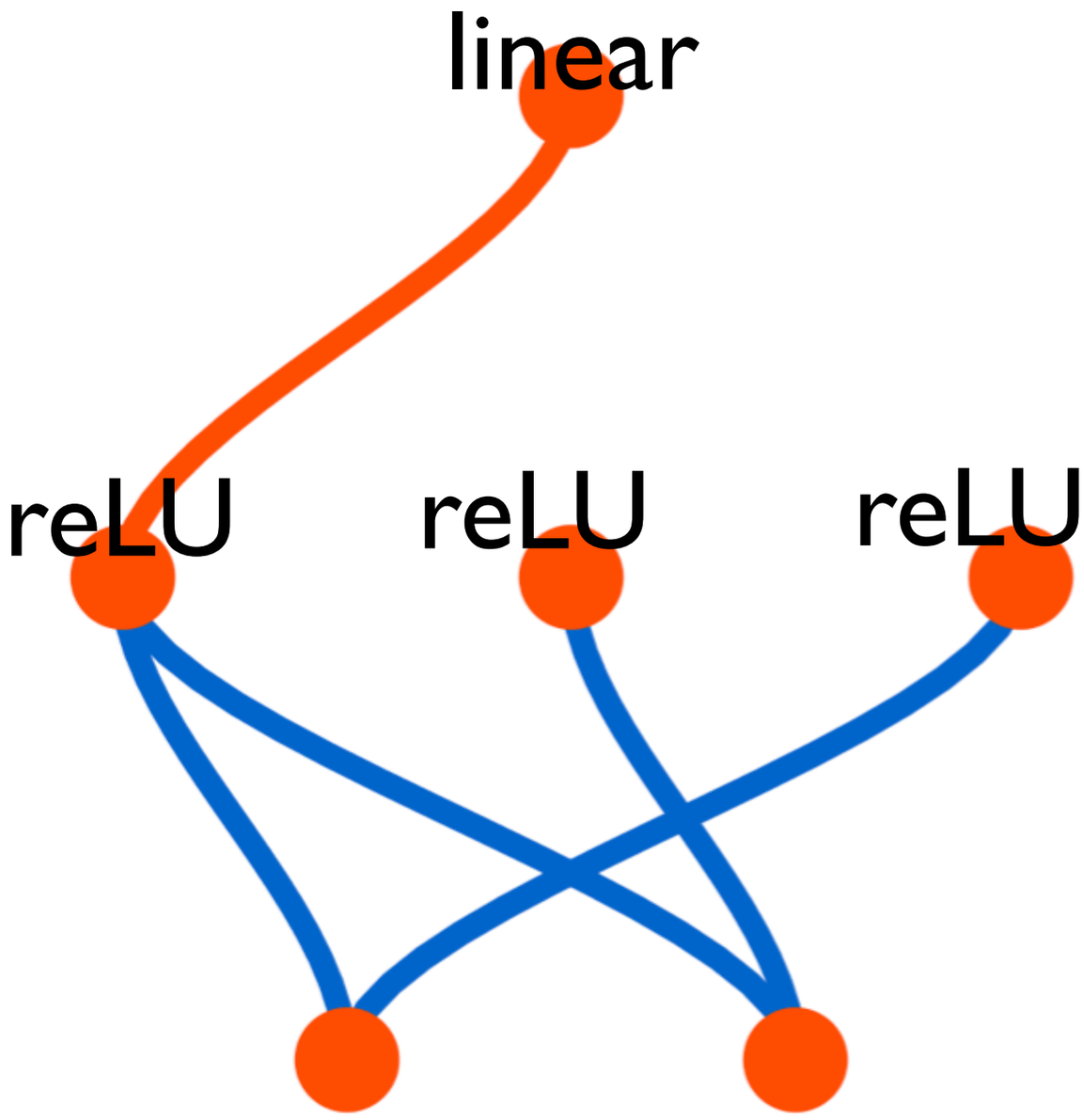

0.0
1.5
3.0
3
2
1
0
−1
−2
−3
y_1
−3
−2
−1
0
1
2
3
y_0

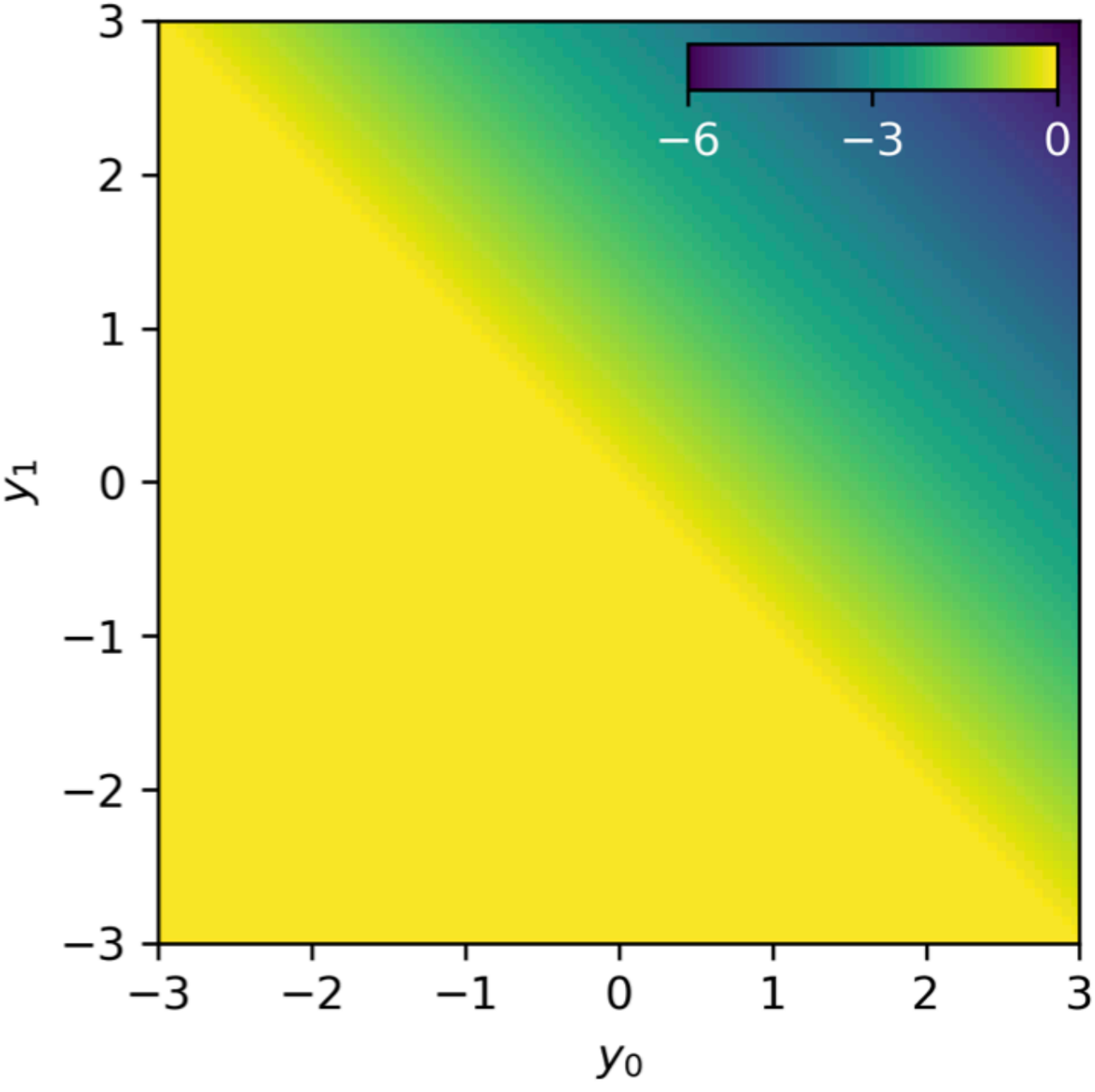

linear
reLU
reLU
reLU

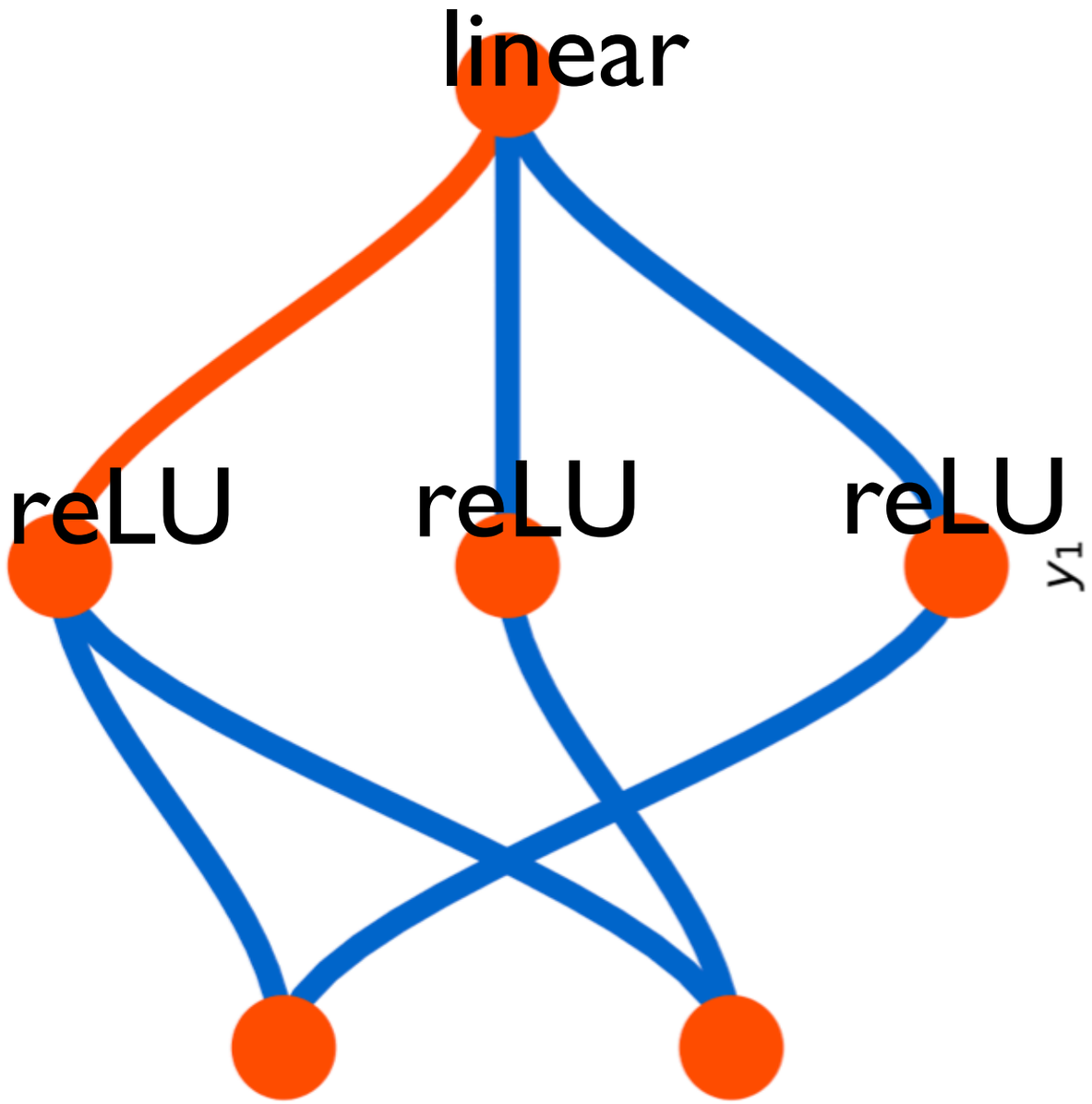

3
2
1
0
−1
−2
−3
y_1
0
3
6
−3
−2
−1
0
1
2
3
y_0

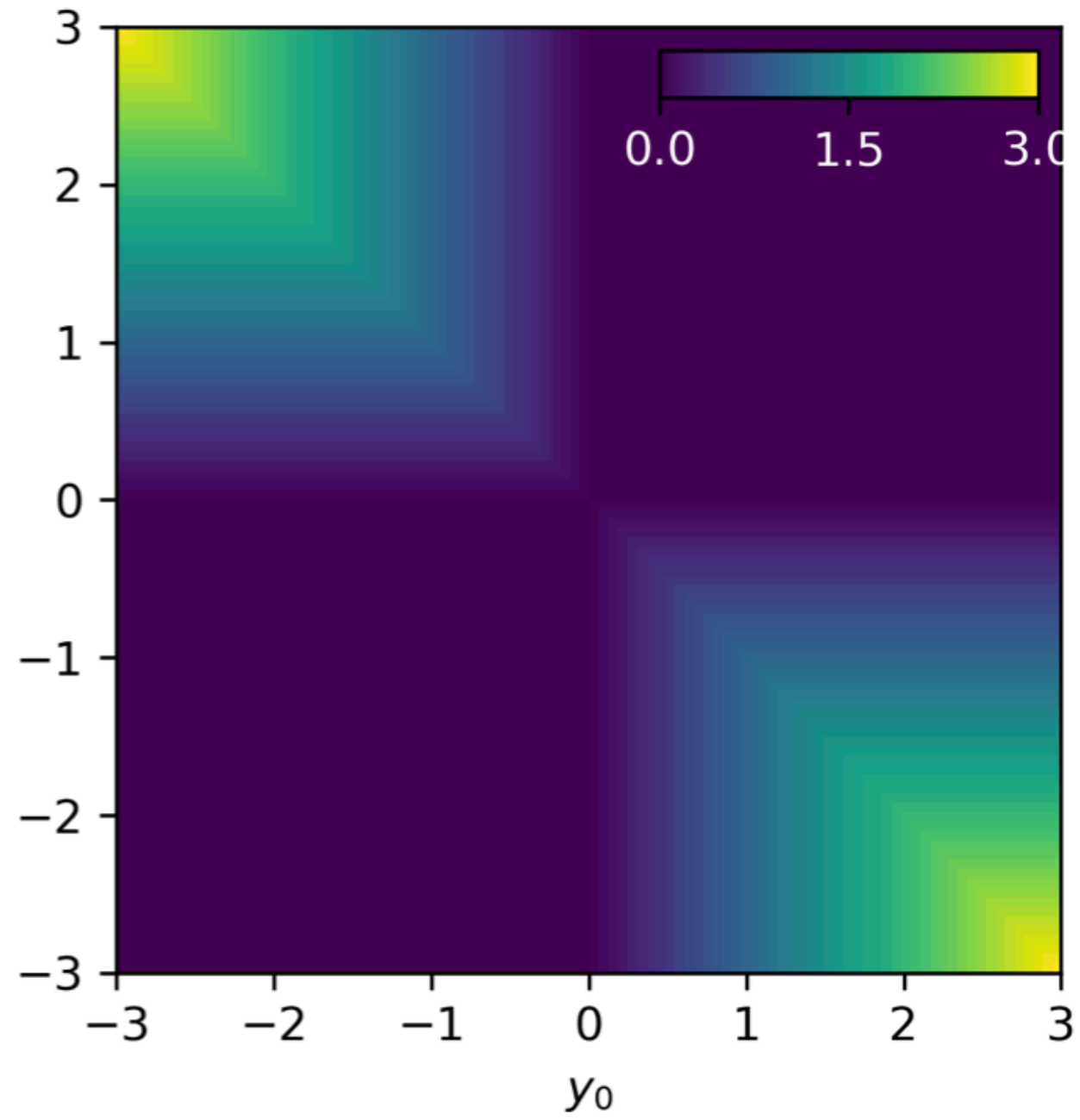

jump
reLU
reLU
reLU

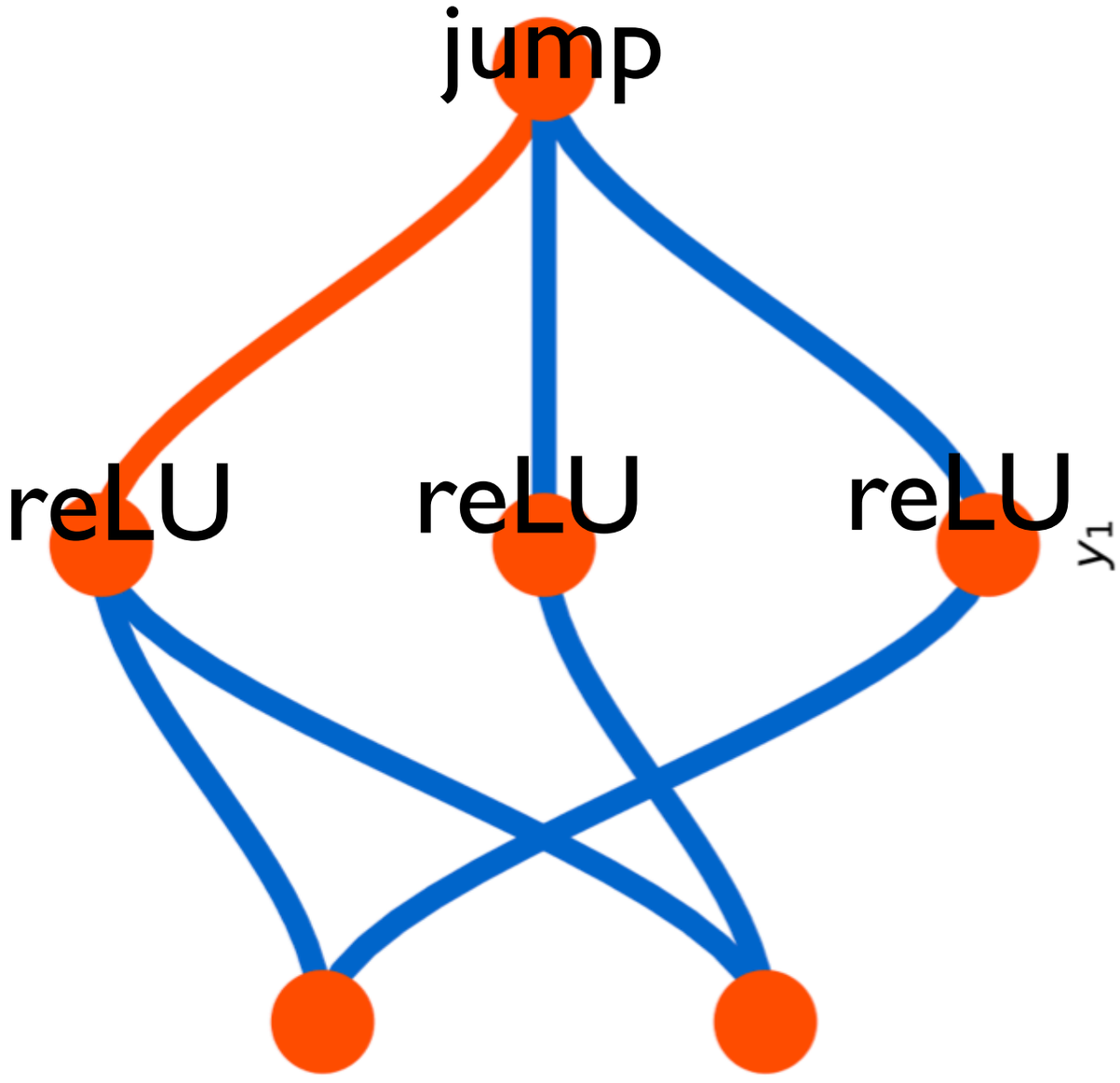

0.0
0.5
1.0
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
y_0
y_1

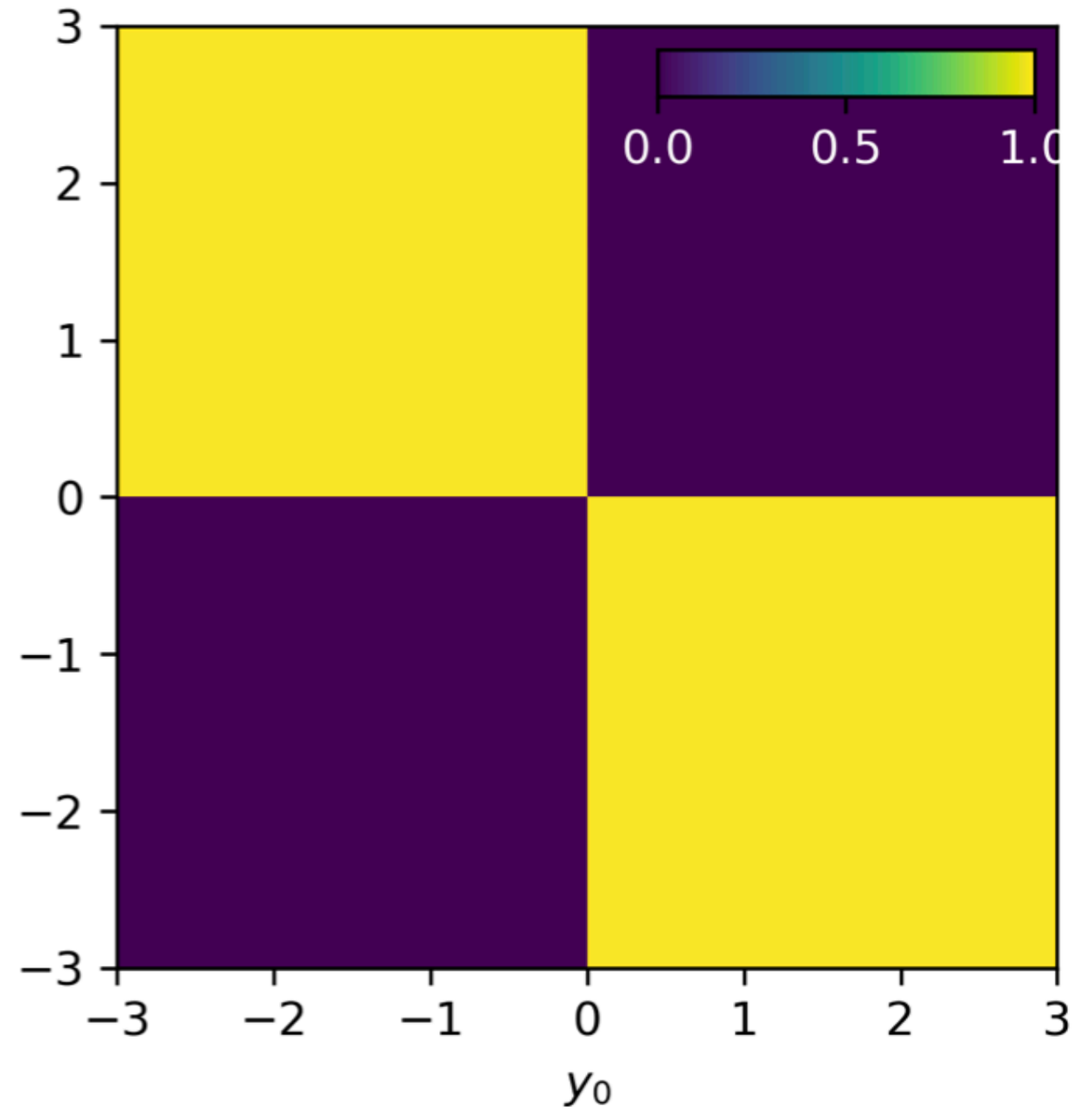

linear
reLU
reLU
reLU


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

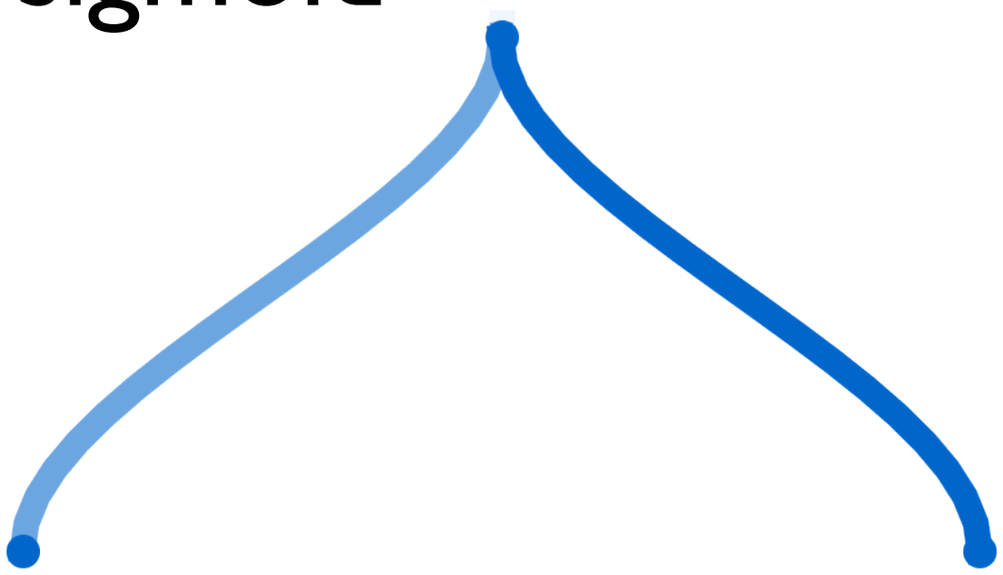

3
2
1
0
−1
−2
−3
y_1
0.0
1.5
3.0
−3
−2
−1
0
1
2
3
y_0

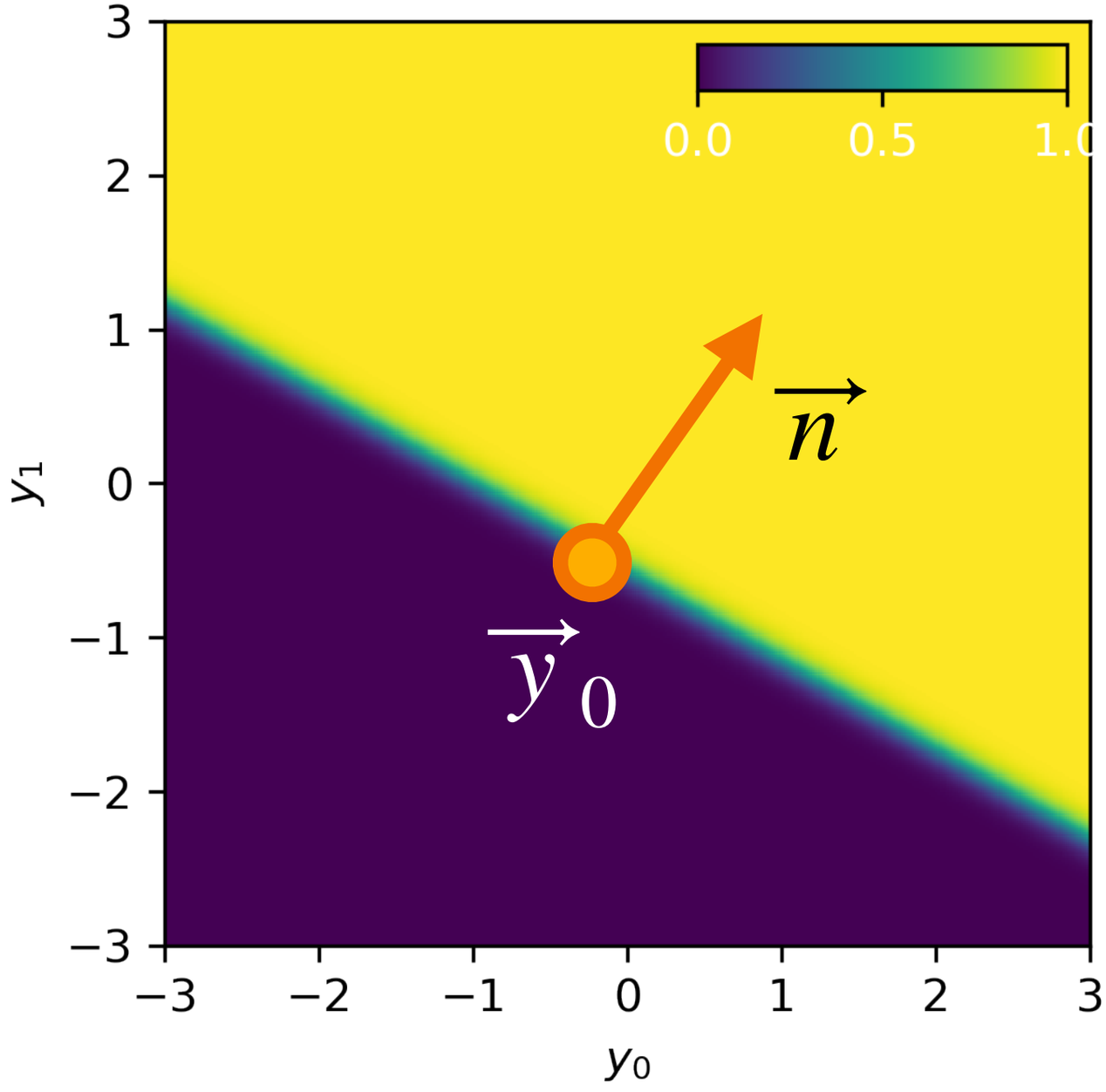

jump
reLU
reLU
reLU


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

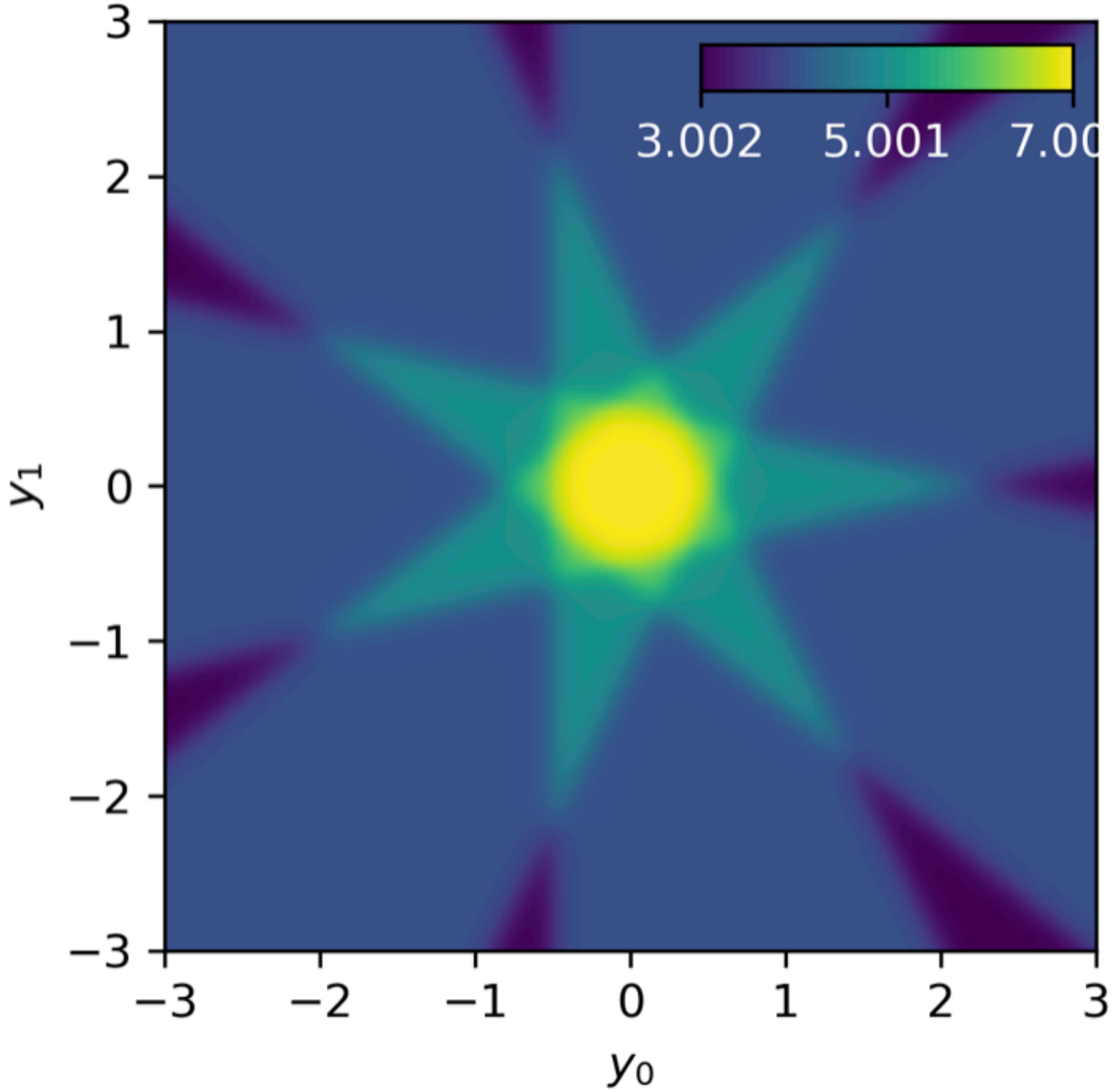




$$y_{\text{out}} = f(\vec{n} \cdot (\vec{y} - \vec{y}_0))$$

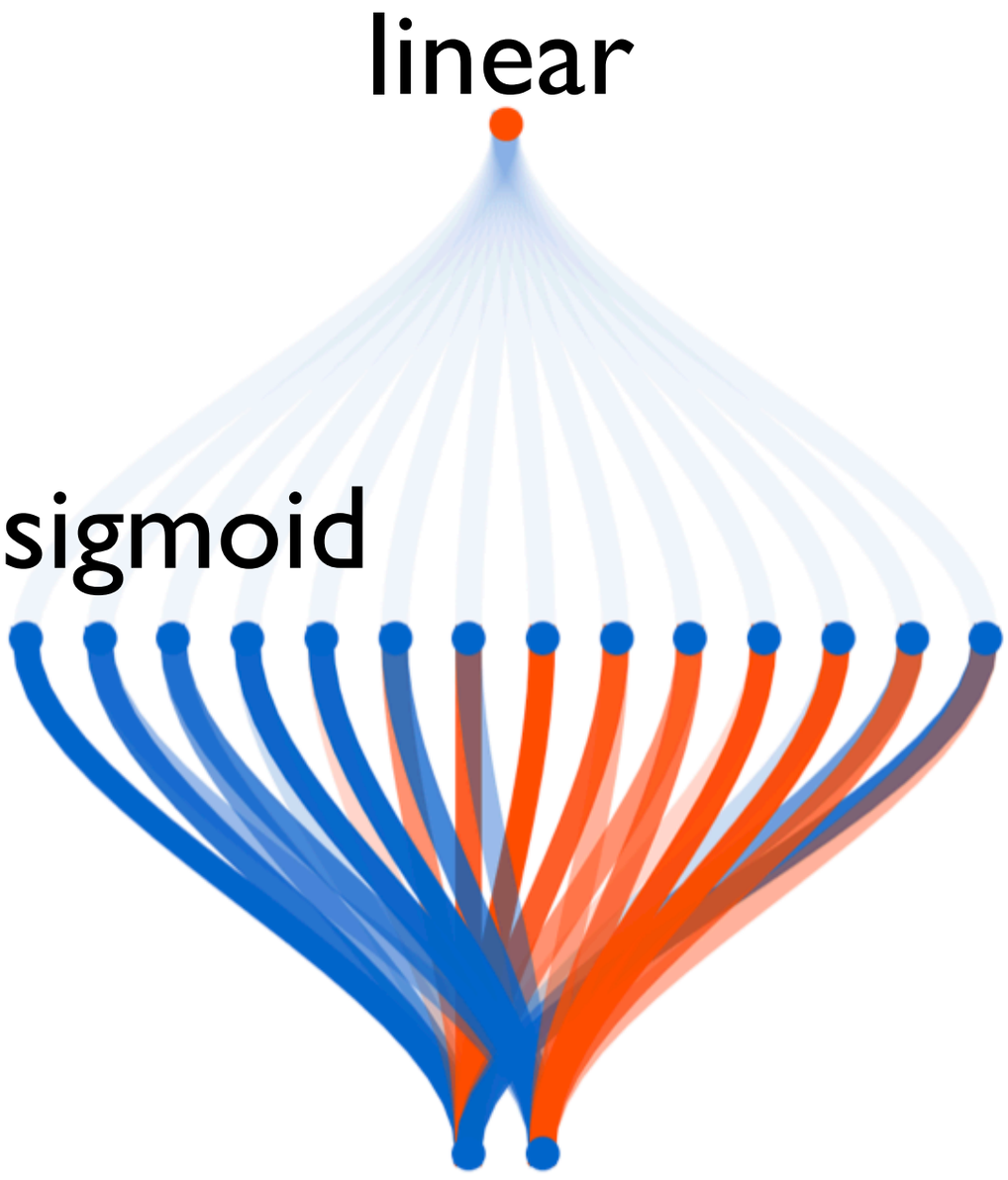
linear

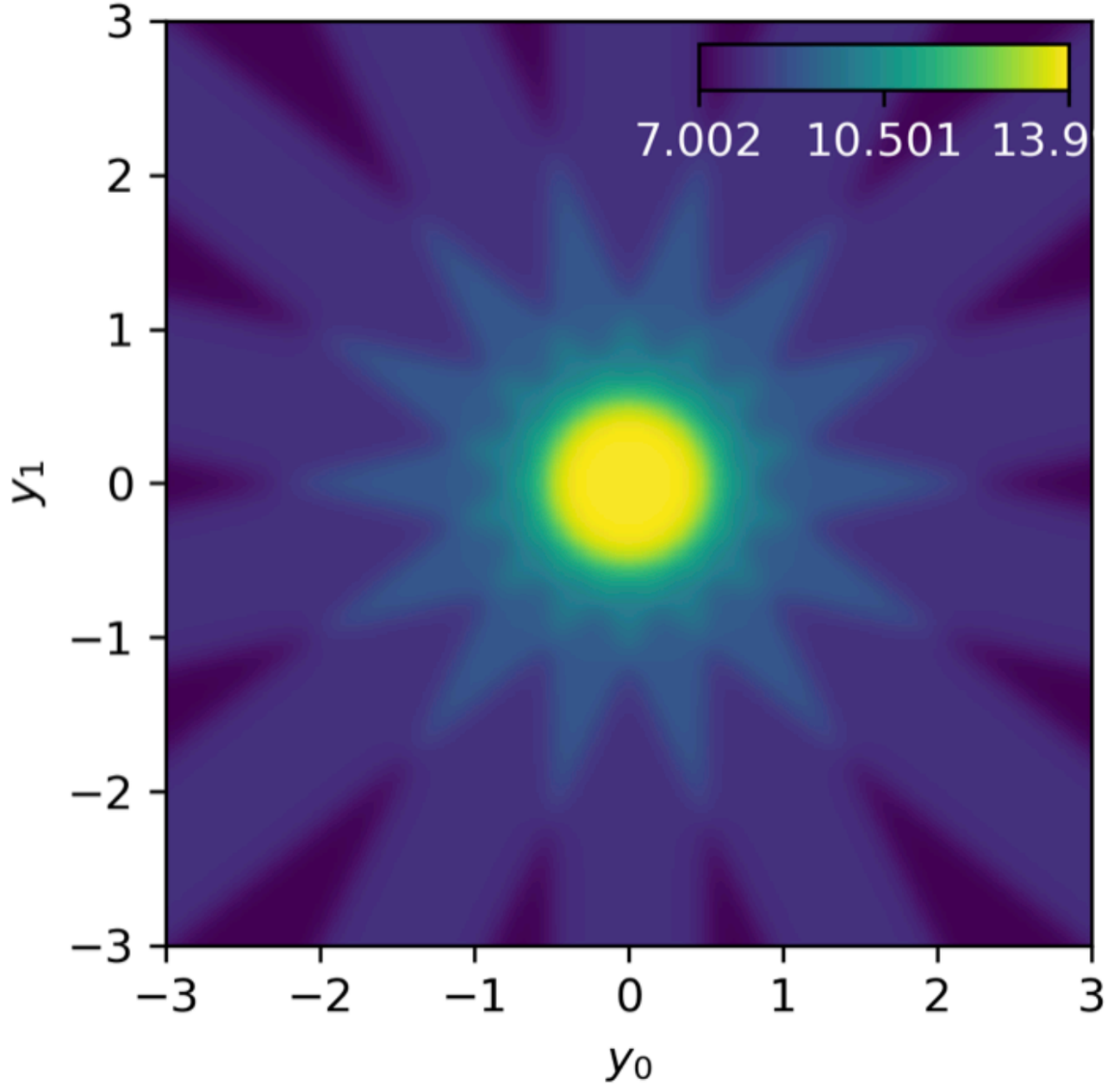
sigmoid

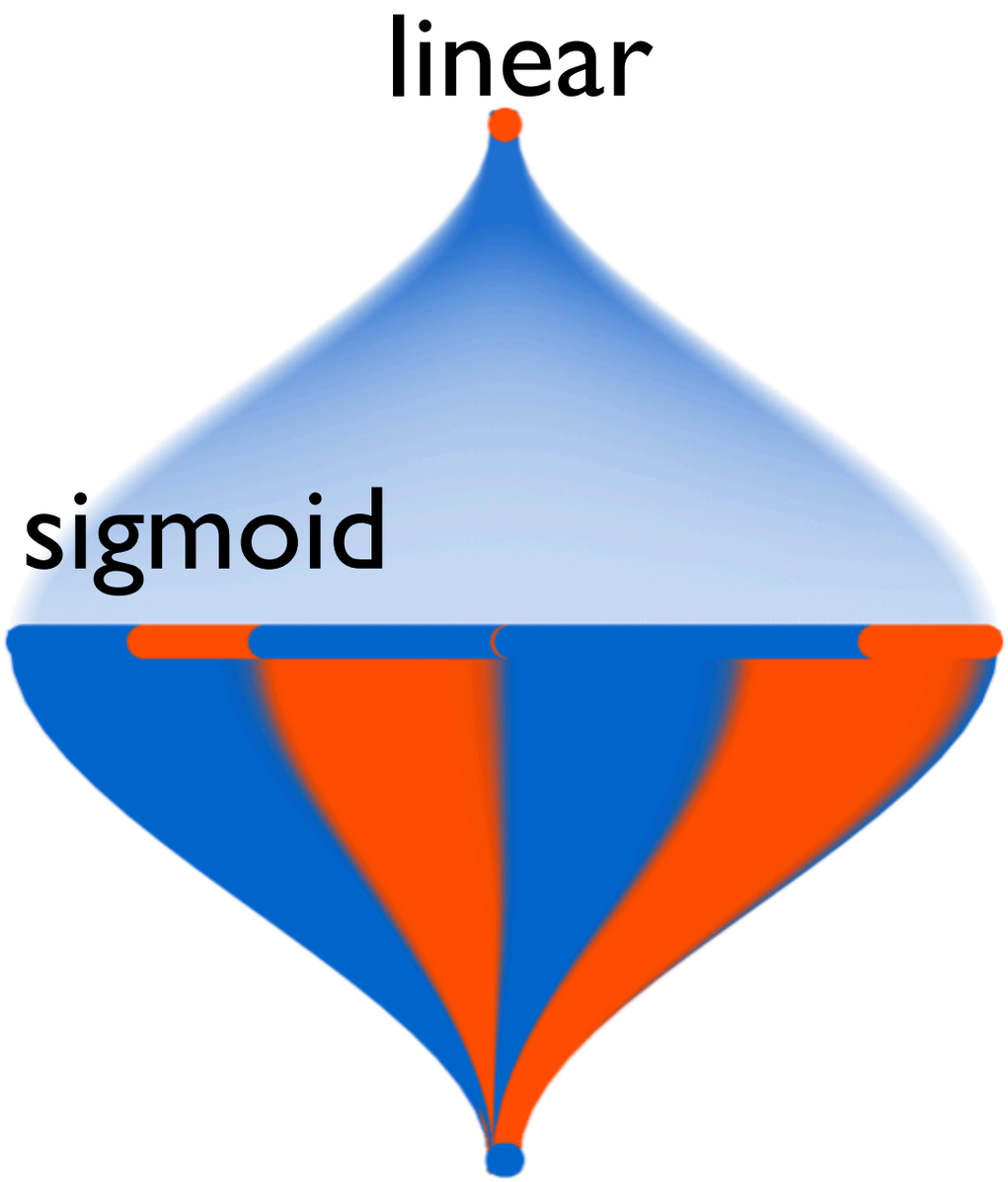


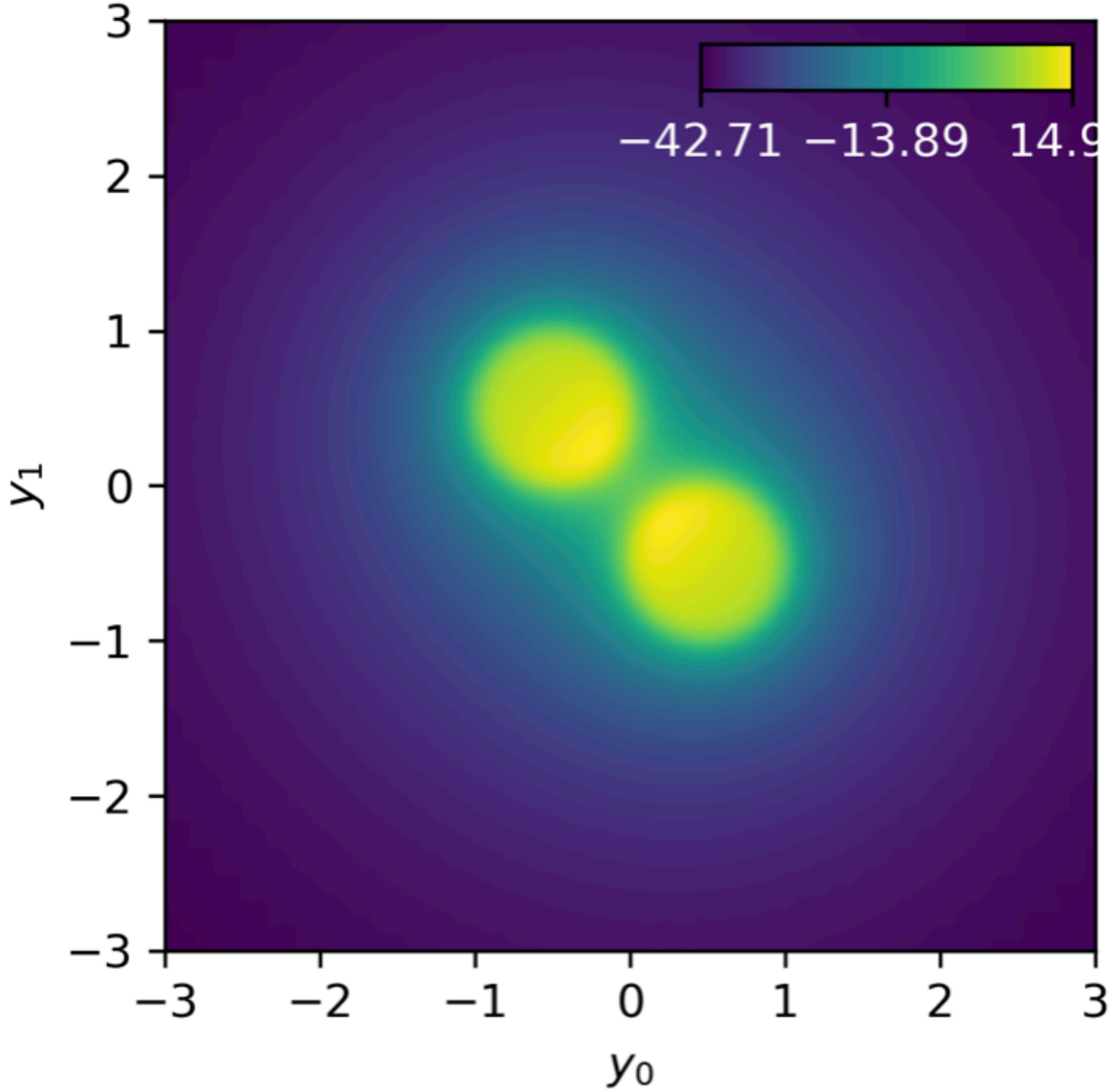
linear

sigmoid



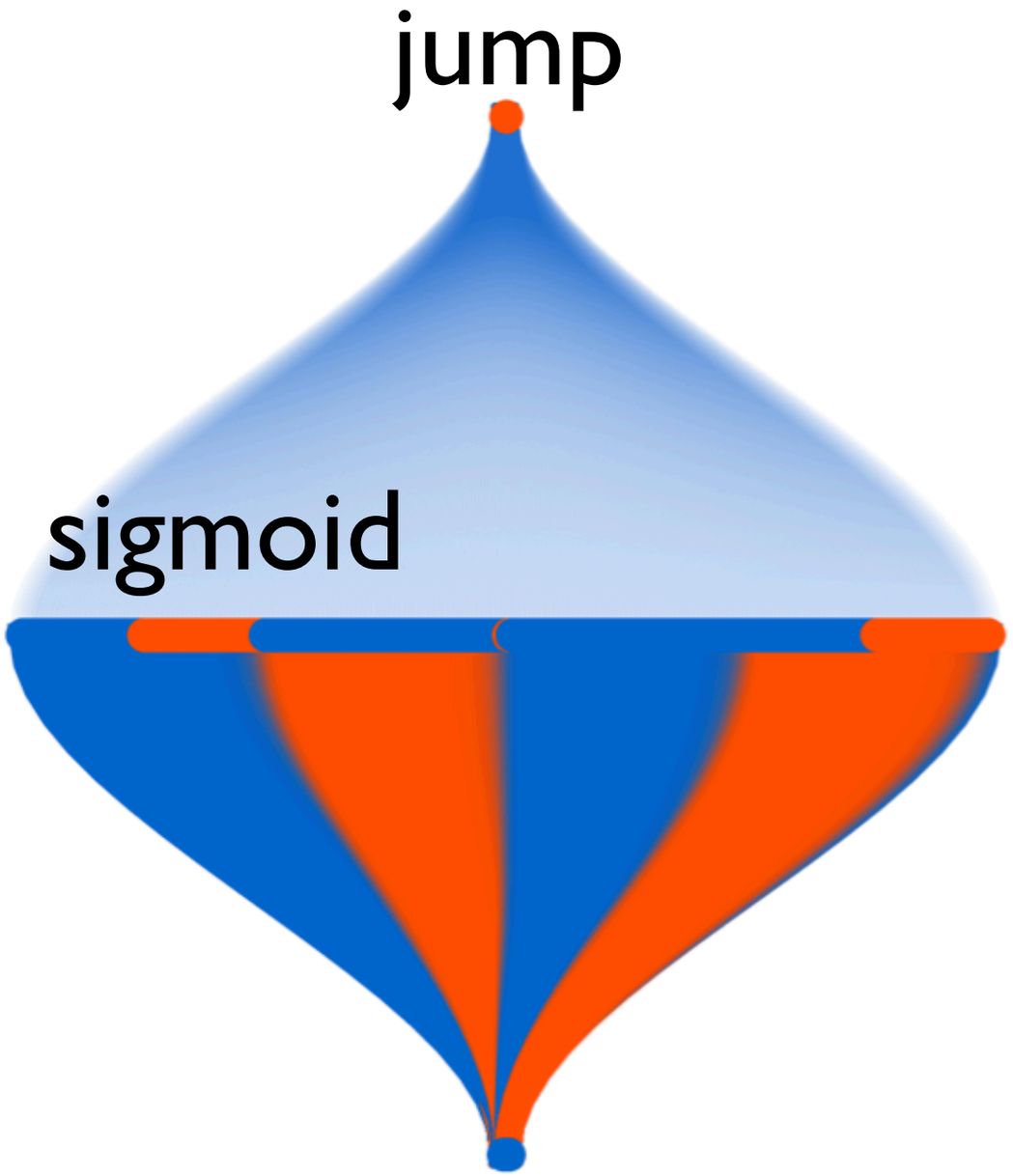


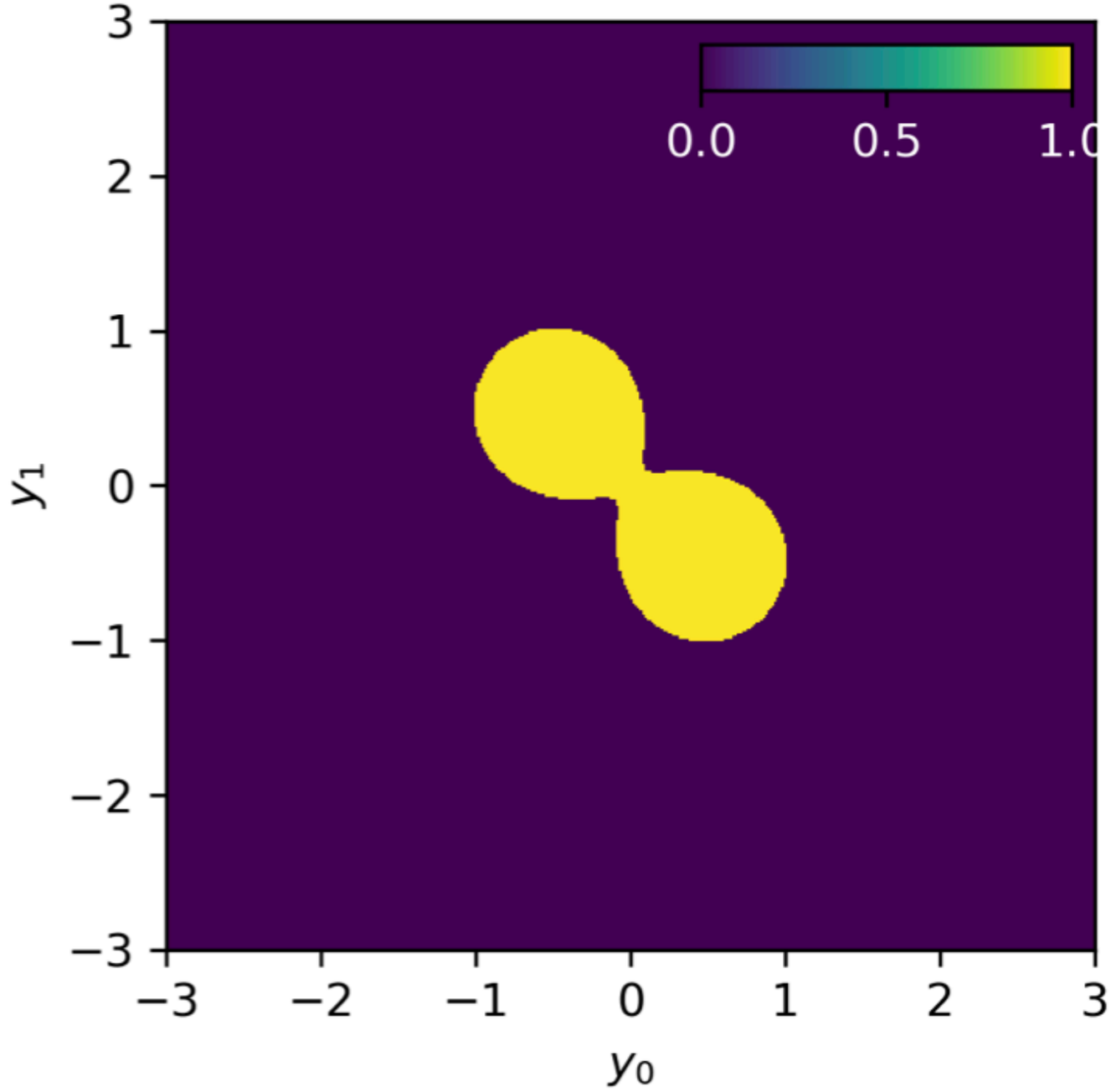




(and we could add more 'blobs')

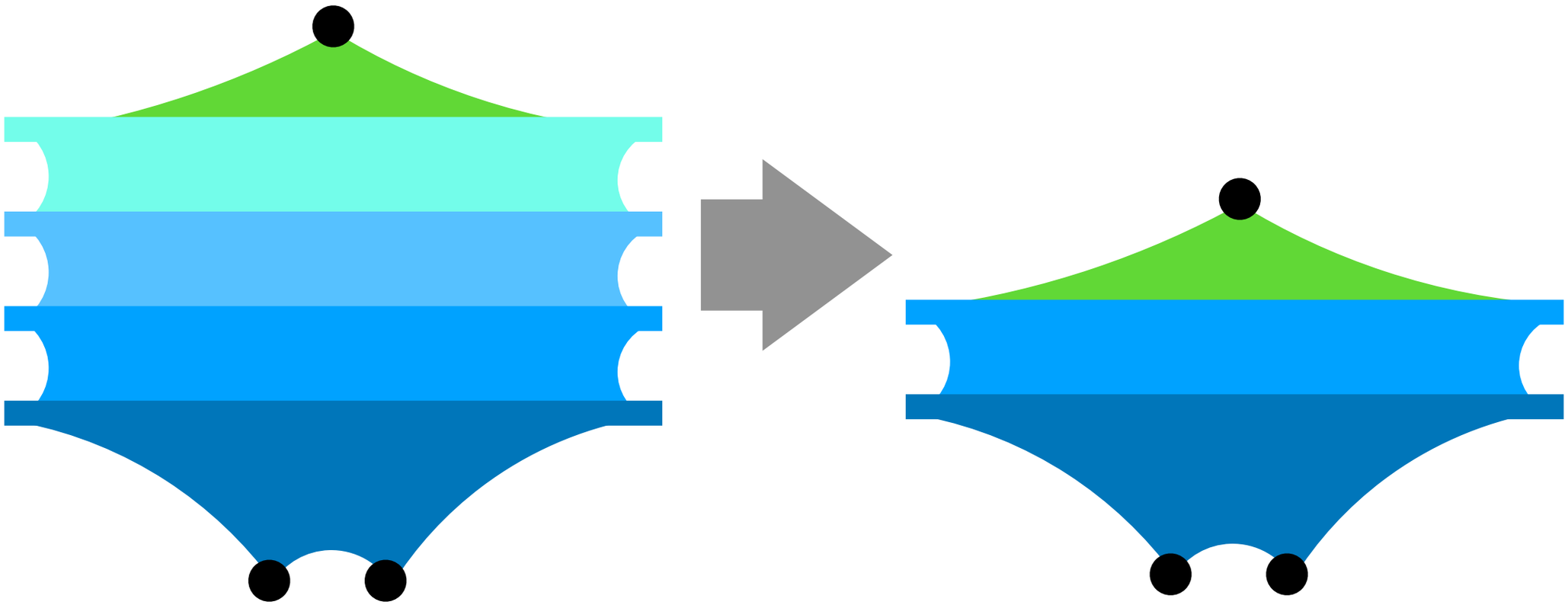
Produce **arbitrary** 2D functions with one hidden layer!


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

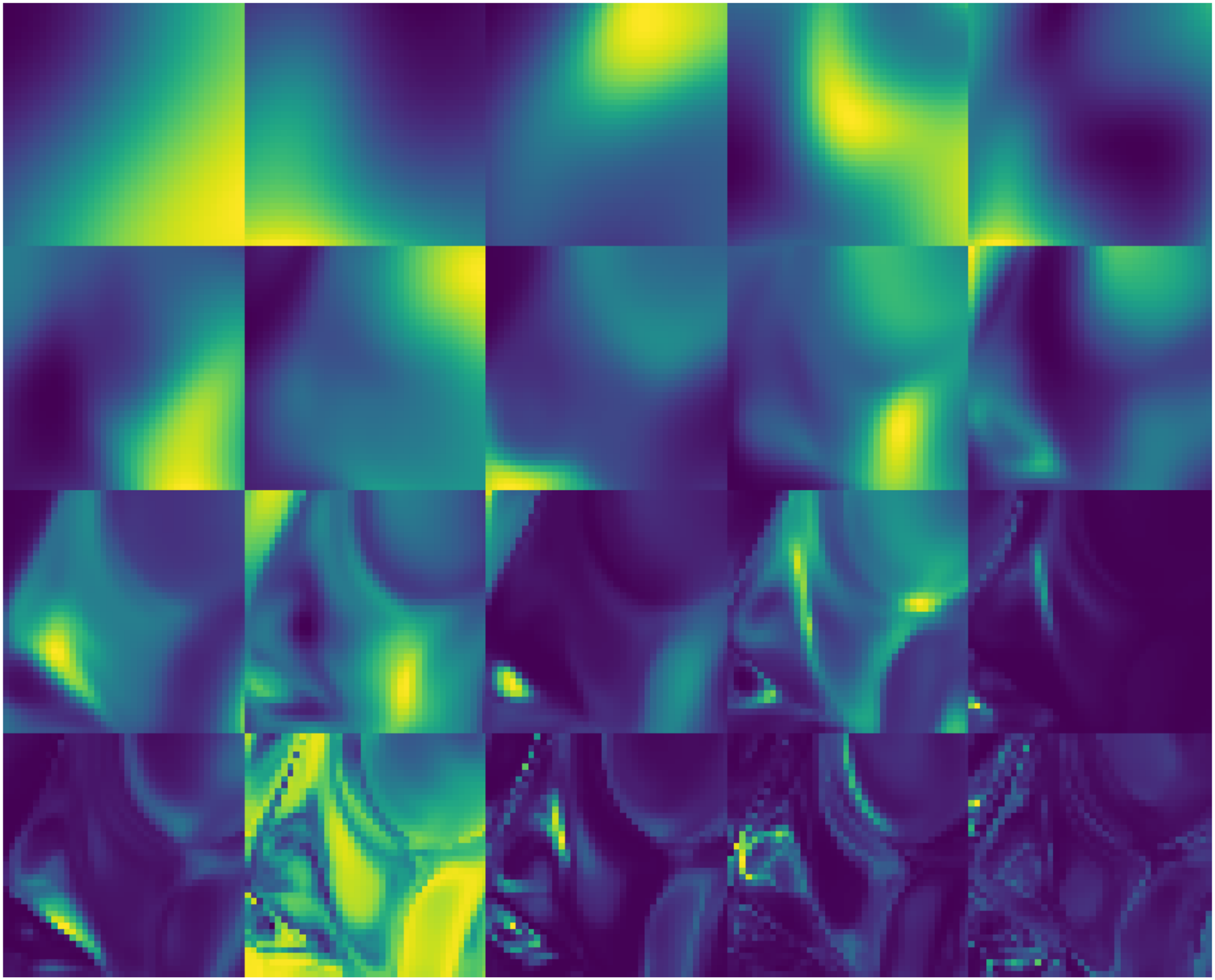
3 Visualize the results of intermediate layers in a multi-layer randomly initialized NN

Visualizing intermediate layers



(cut away higher layers)

Visualizing intermediate layers

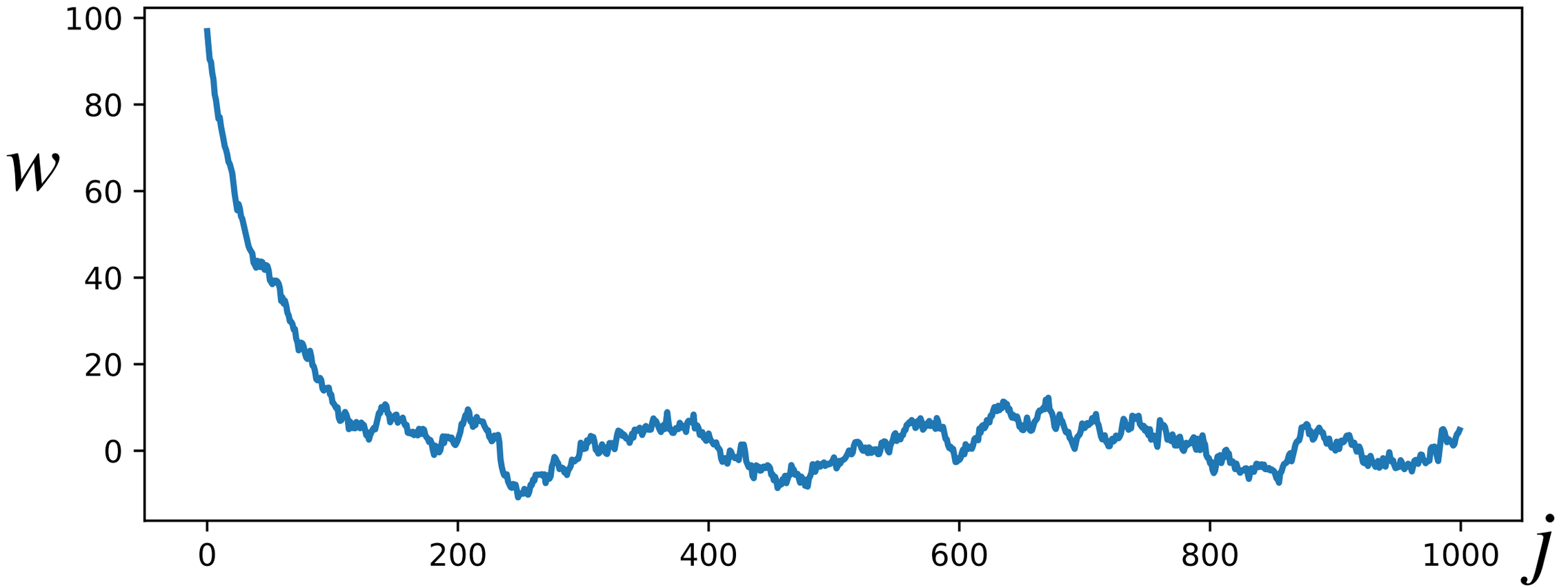


4. What happens when you change the spread of the random weights?
5. Explore cases of curve fitting where there are several (non-equivalent) local minima. Is sampling noise helpful?

Quiz

Machine Learning for Physicists

Stochastic gradient descent



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

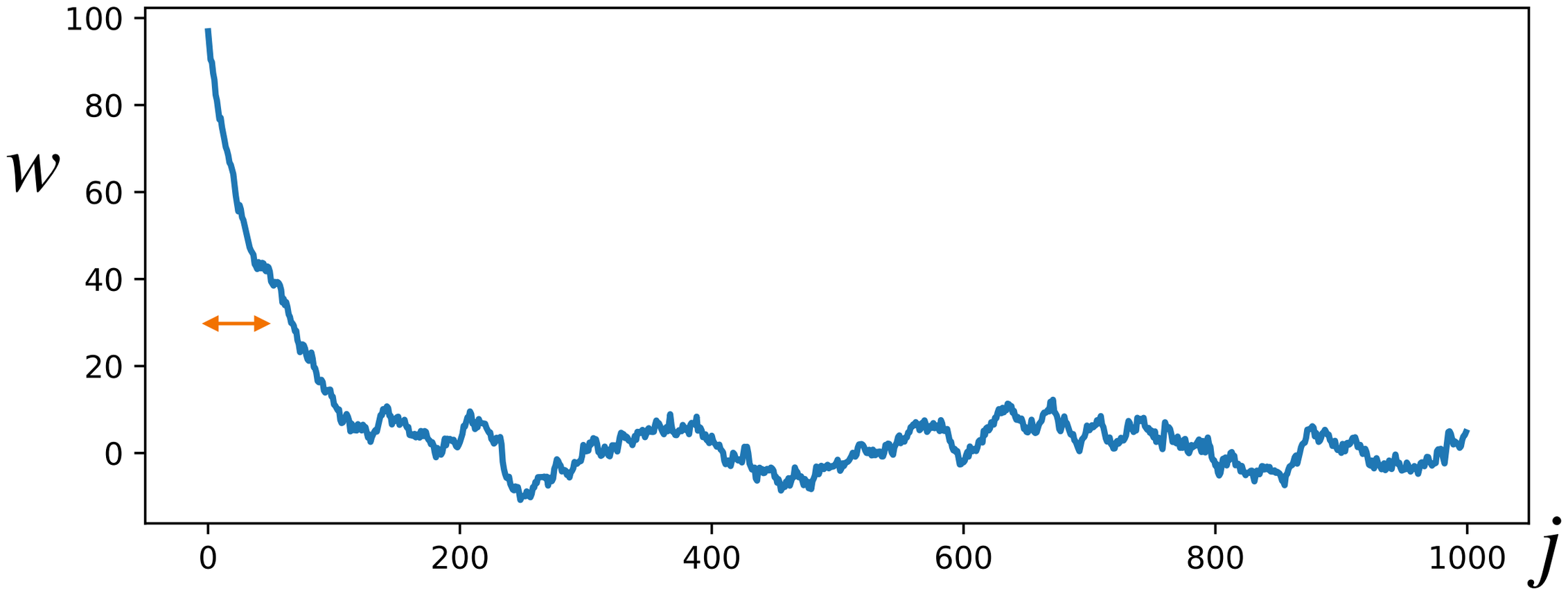
What is the value of η ?

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 η ?

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

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

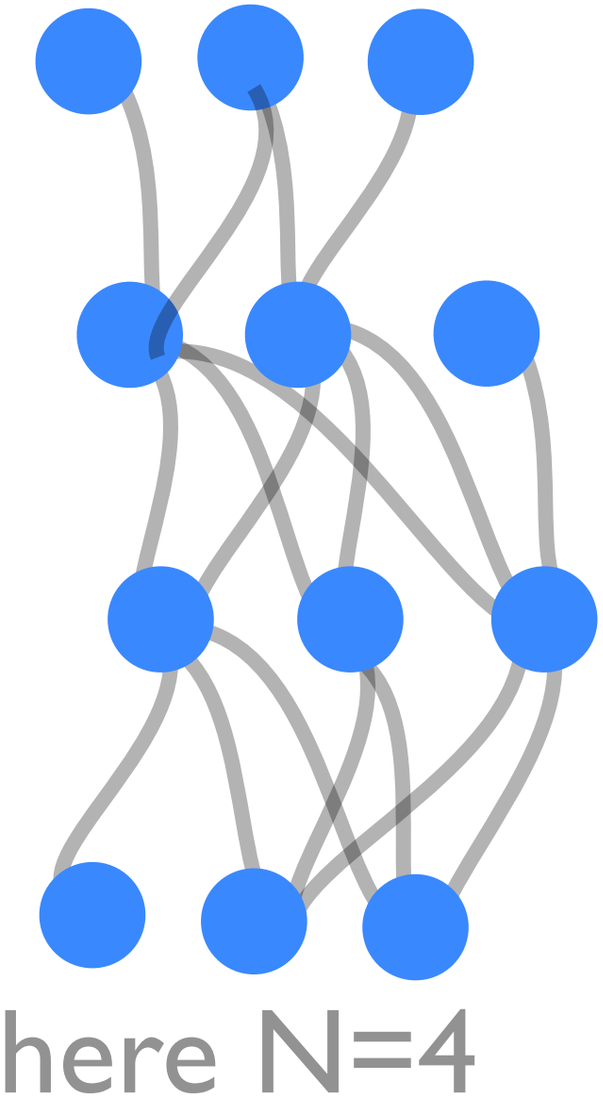
$$\frac{\partial z}{\partial w_2} =$$

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:

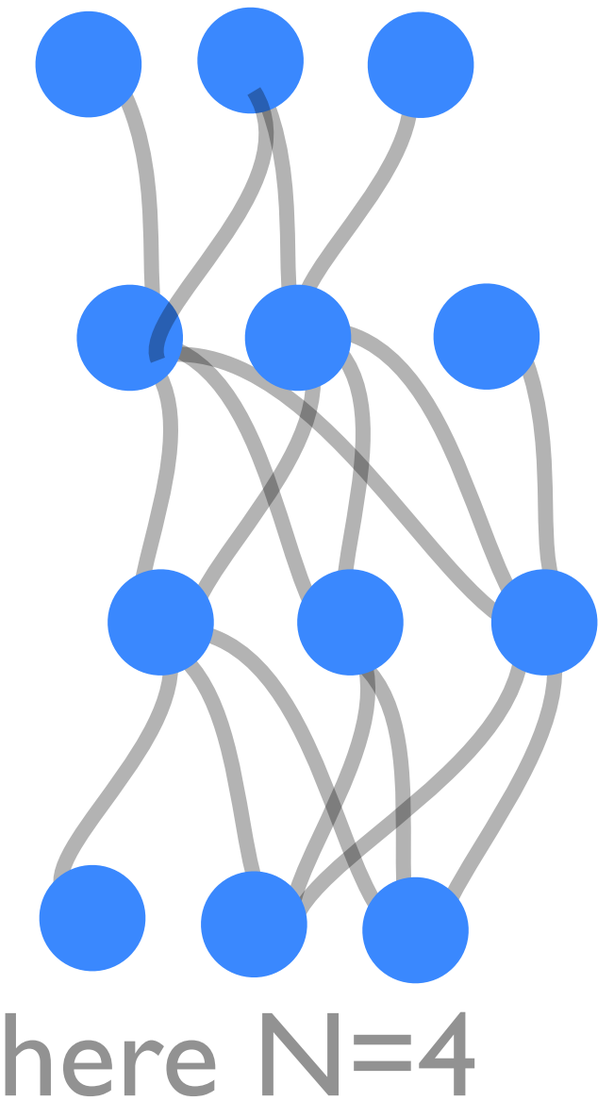
What would be the amount of network evaluations required if one tries to deduce all the derivatives of the (sample-averaged) cost function with respect to the weights brute-force numerically?

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)

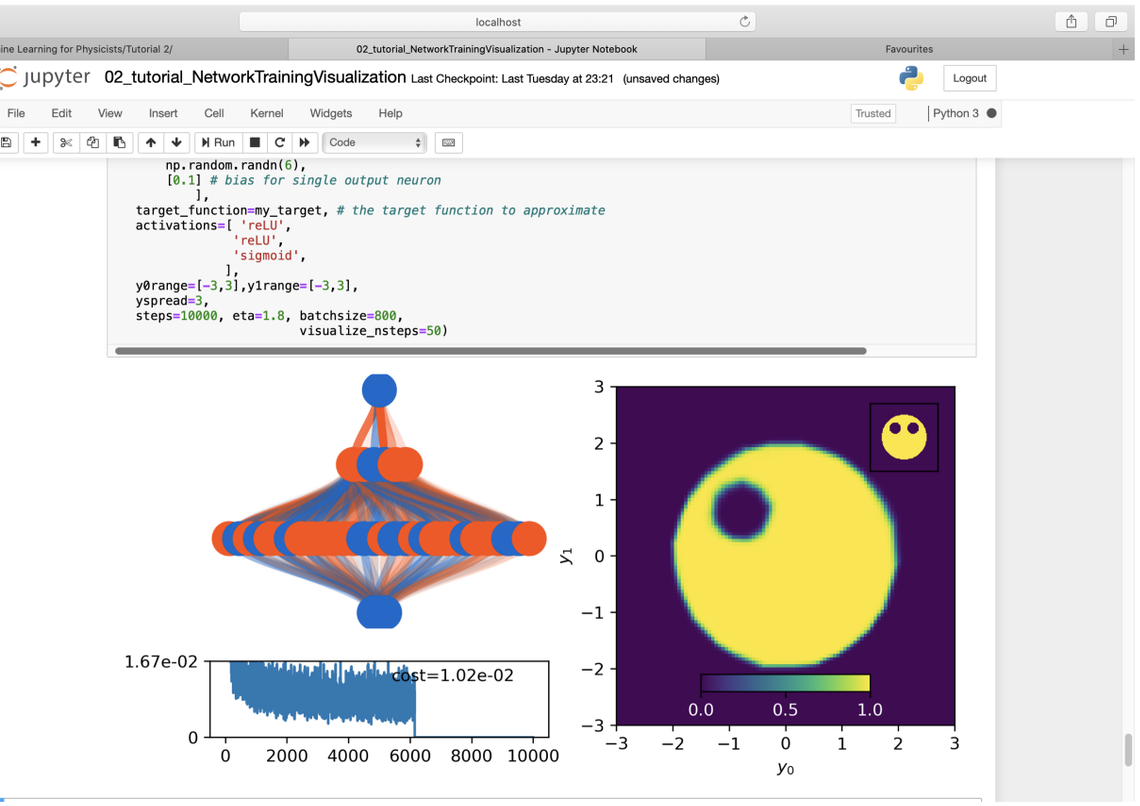
Machine Learning for Physicists

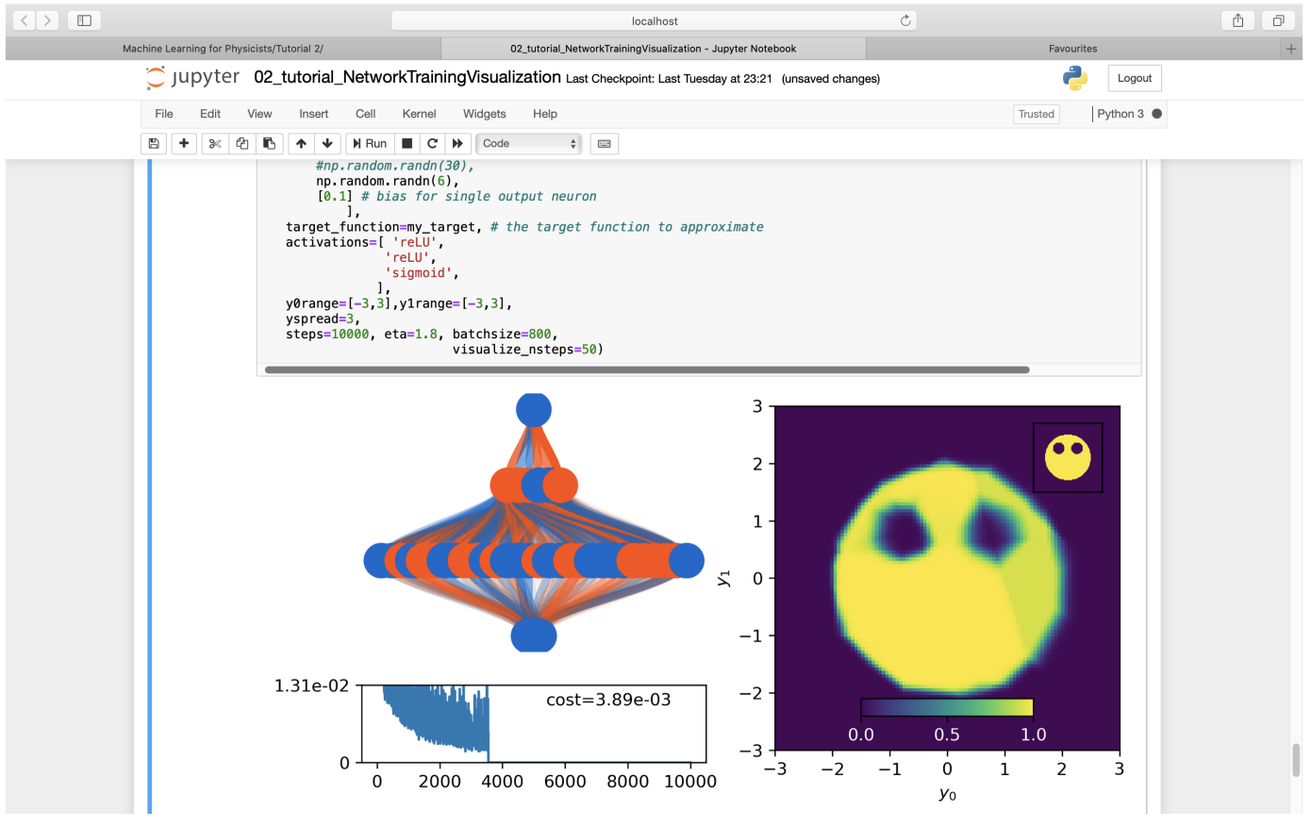
1* Carefully study the backpropagation algorithm, on paper and in the program

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

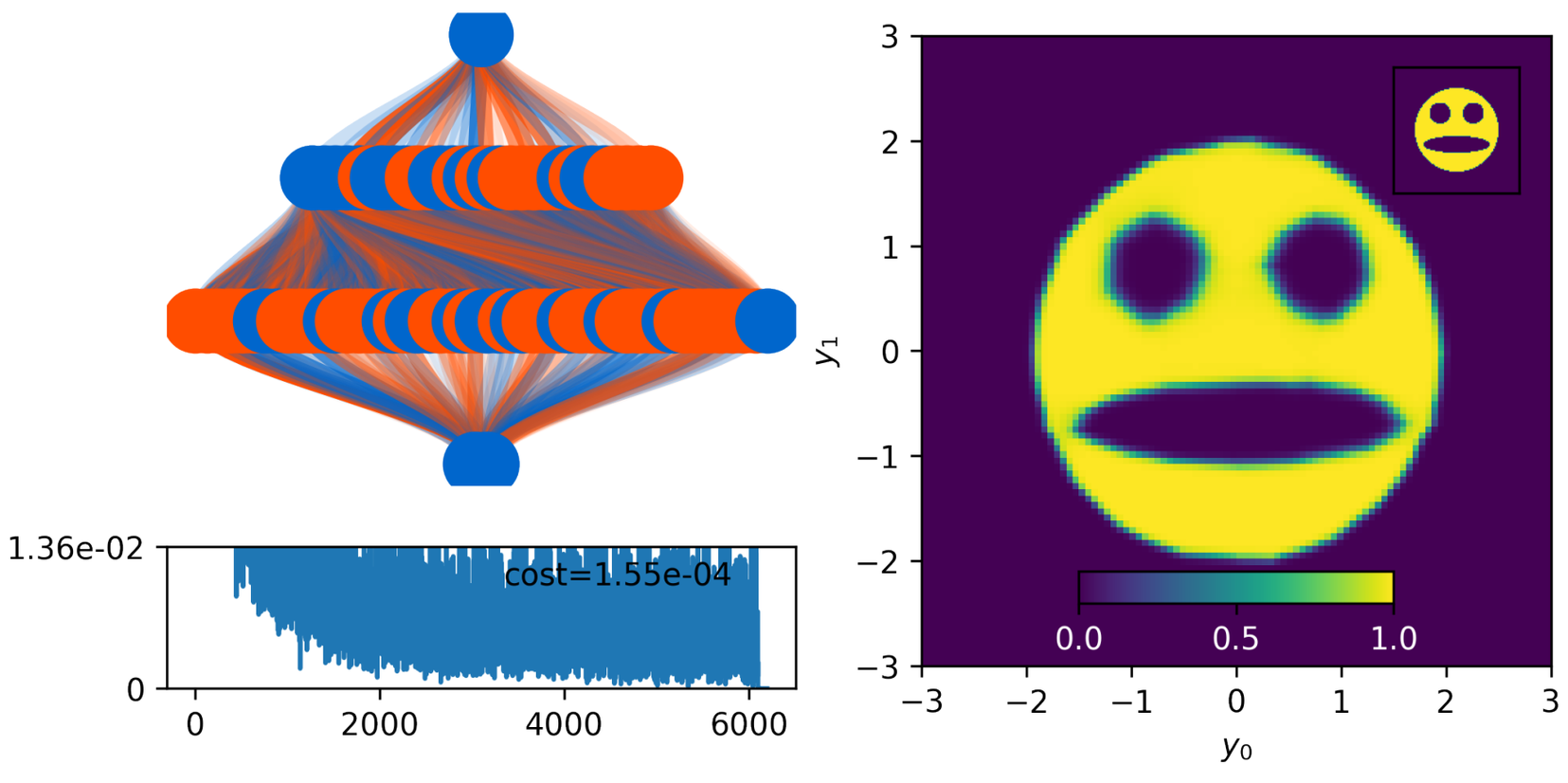
* minimum

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



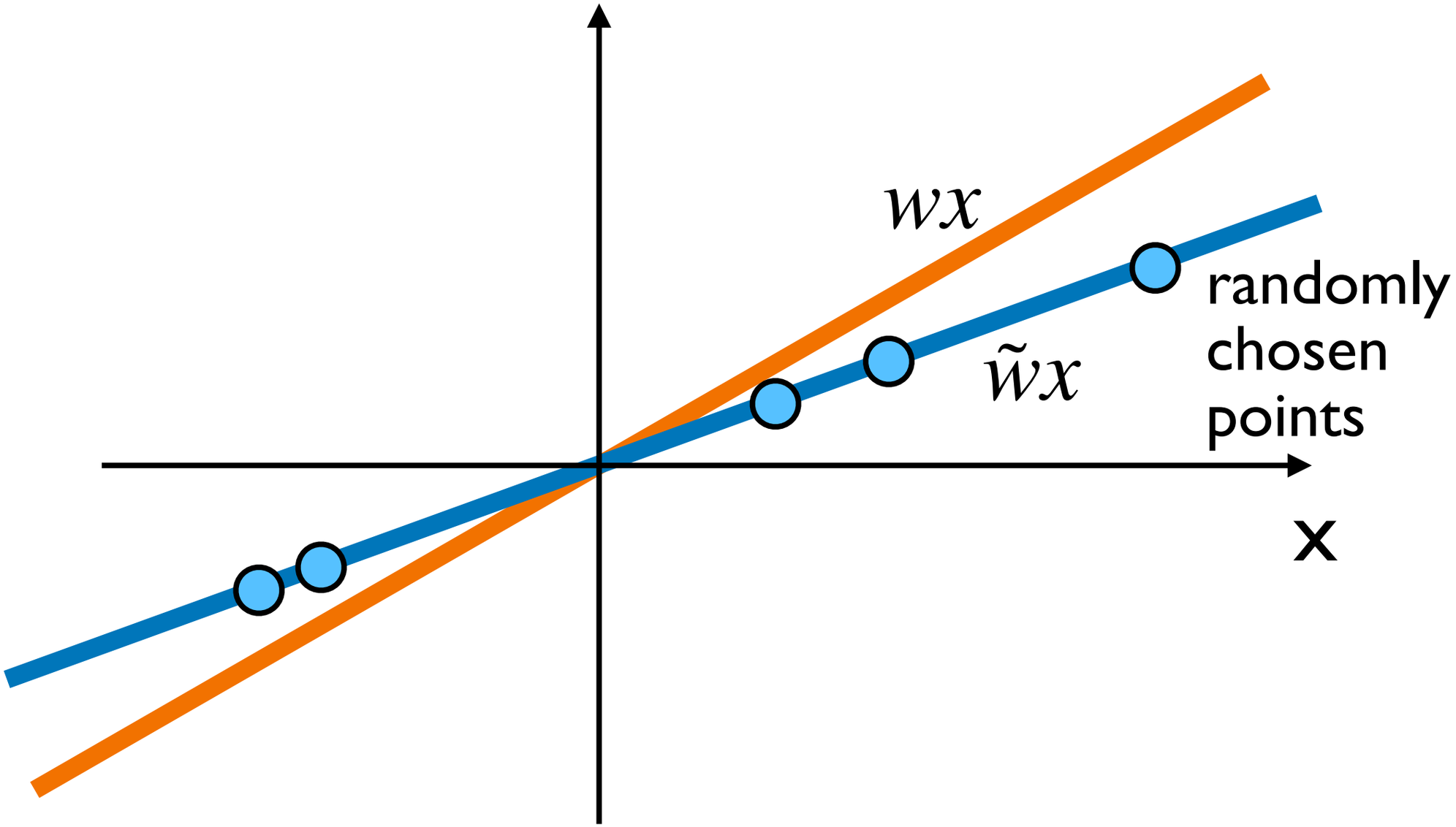


(Anirudh)



(Darshan)

Linear fitting… ("least squares")



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

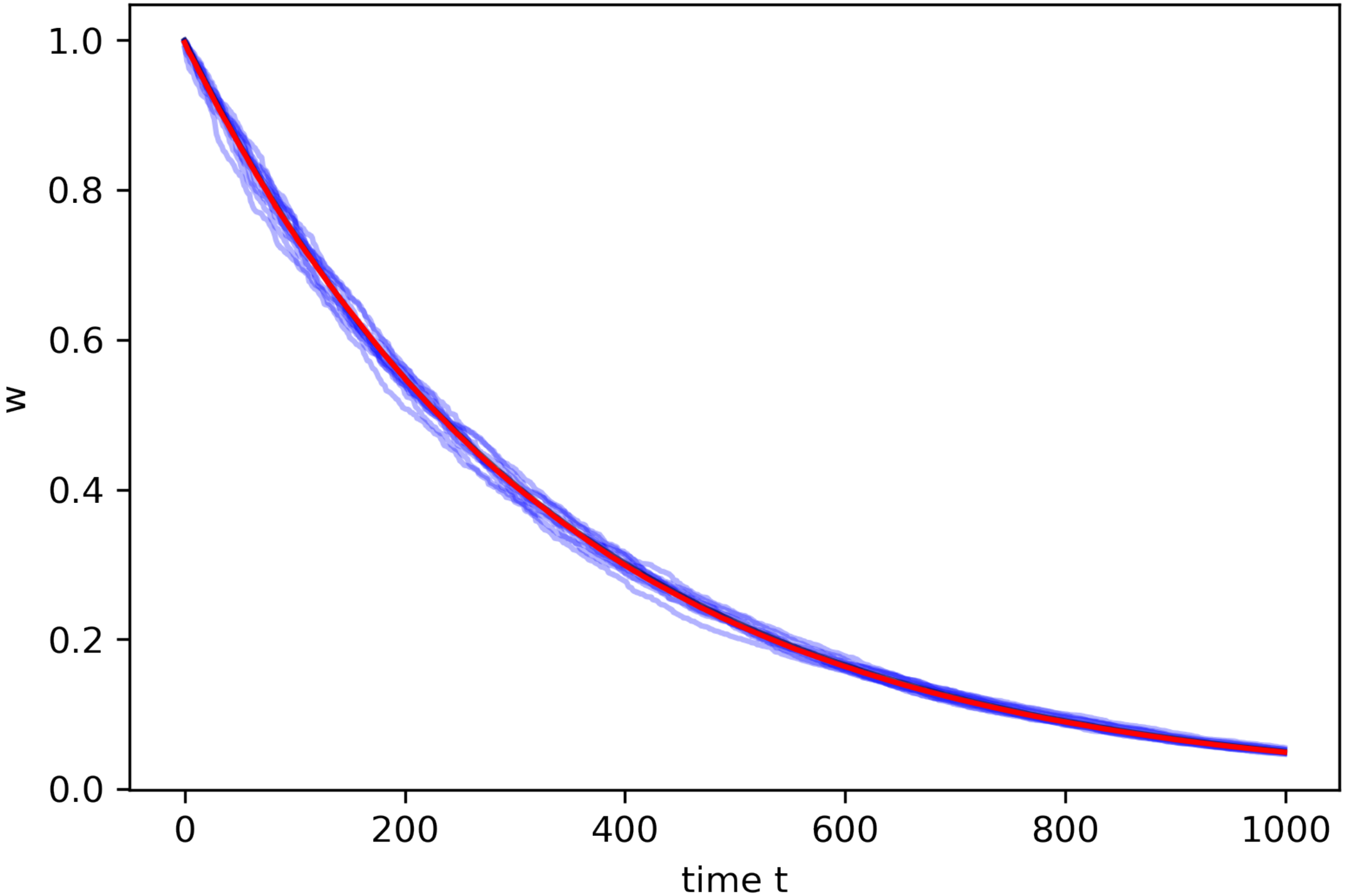
$$C = \frac{1}{2} \sum_{j=1}^{N} (w - \tilde{w})^2 x_j^2$$

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

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

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

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


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

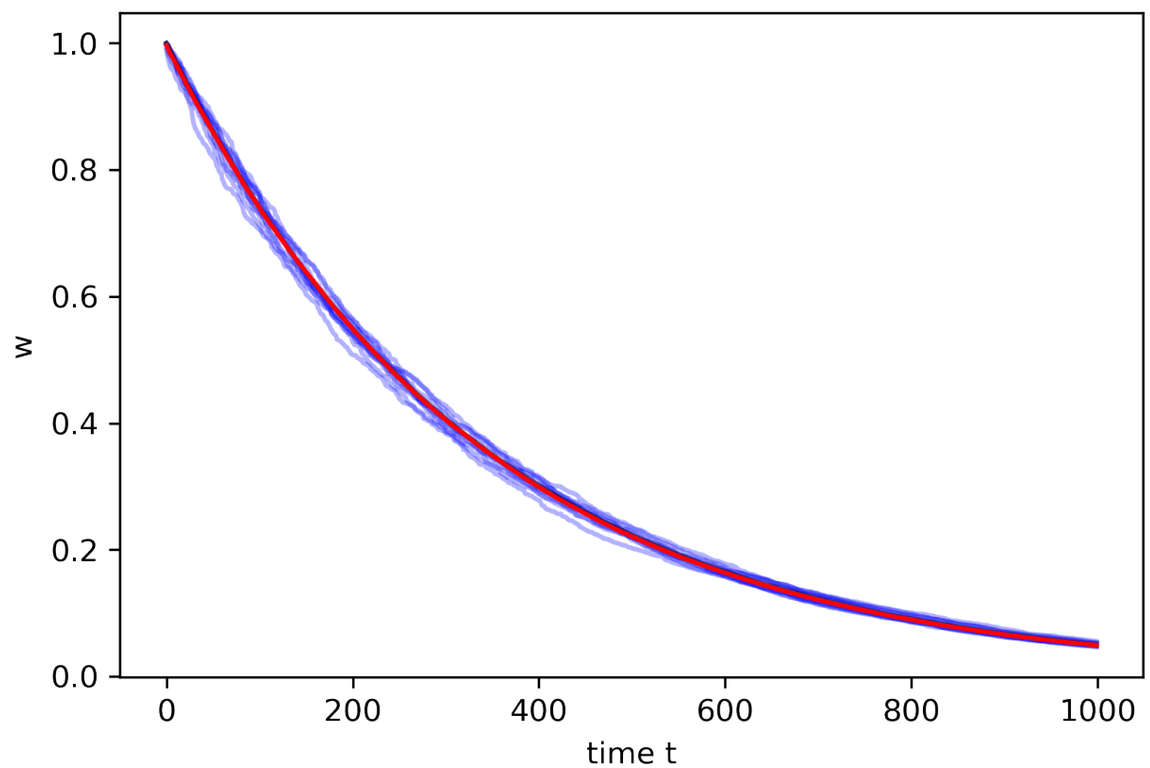
then: $S = N + \sqrt{2N}\xi$

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

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

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

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)

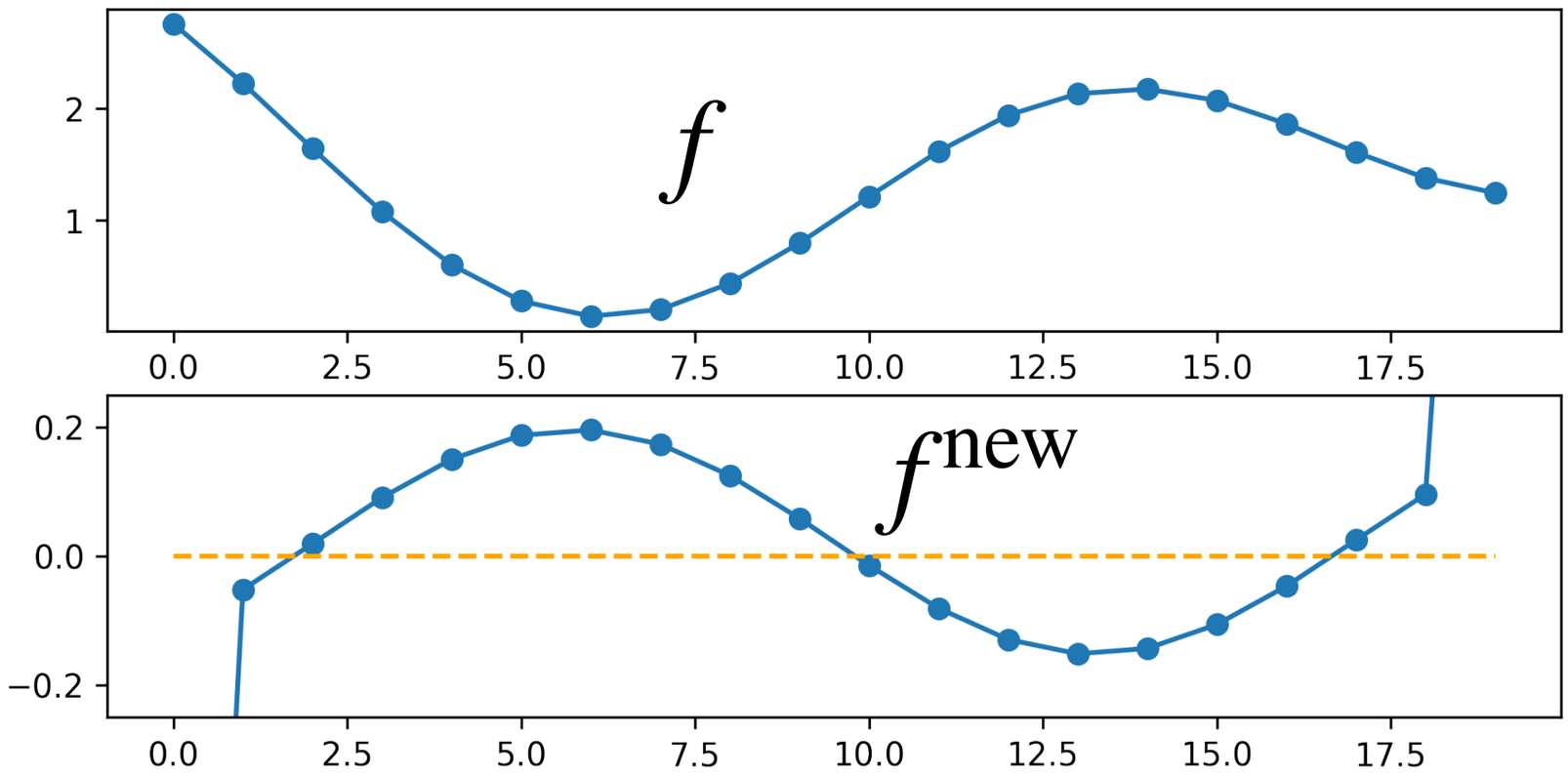
3 Produce a simple network with keras that can distinguish a 2D picture of a randomly placed square from that of a randomly placed circle! What happens if you add noise to the images?

* minimum

Quiz

Machine Learning for Physicists

$$f_j^{\text{new}} = \sum_n K_n f_{j+n}$$

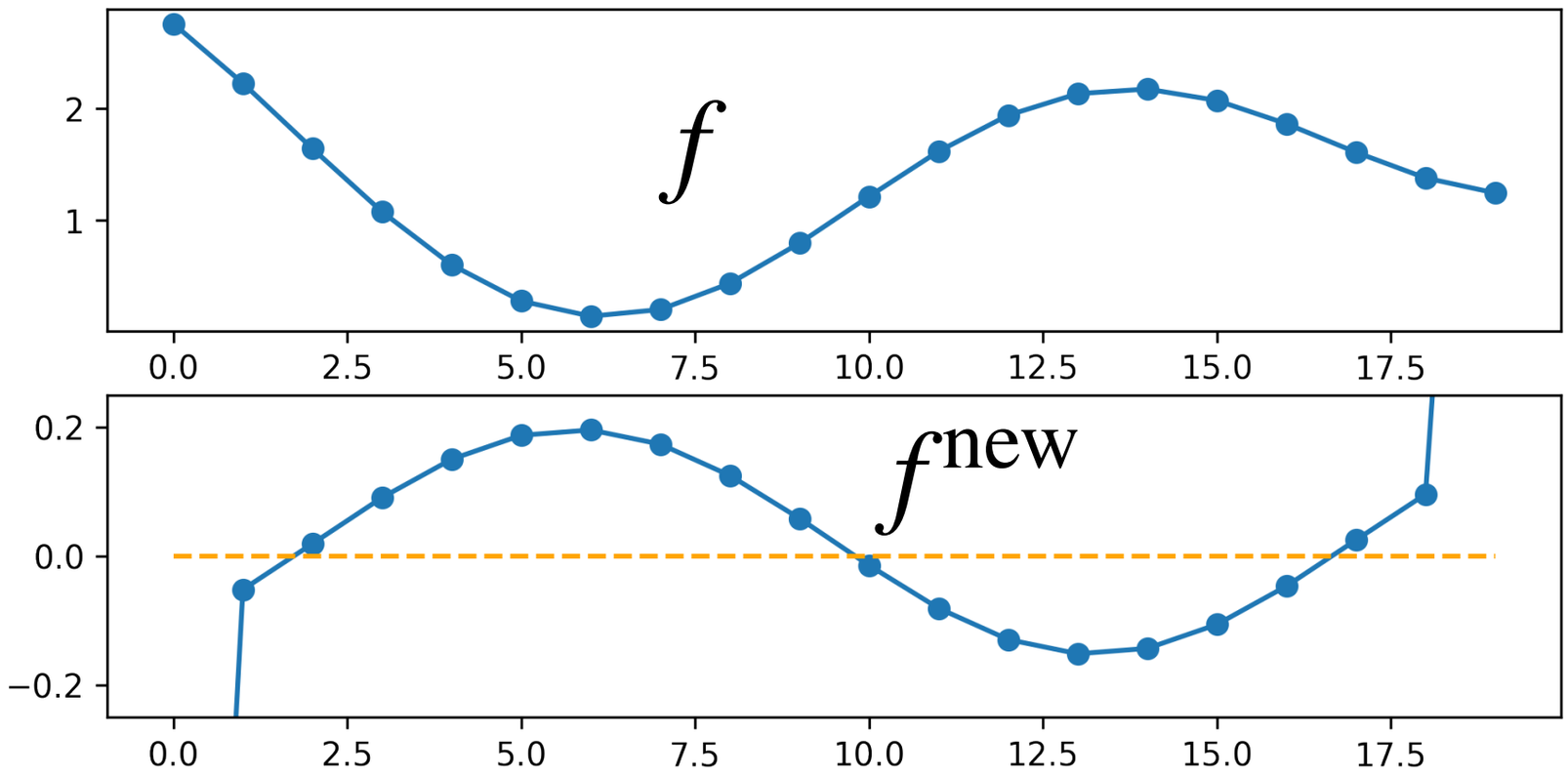


What was the kernel K?

a [-2,0,2] **c** [-1,0,1] **e** [1,2,1]

b [-1,2,-1] **d** [1,-2,1] **f** [1,0,-1]

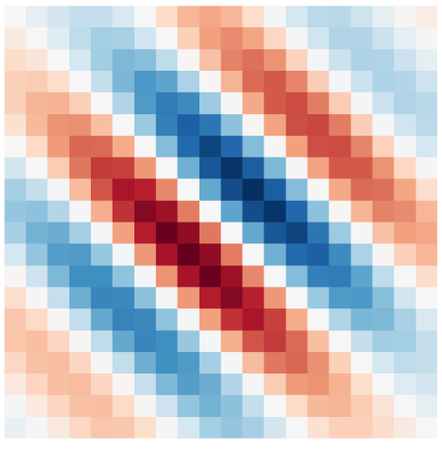
$$f_j^{\text{new}} = \sum_n K_n f_{j+n}$$



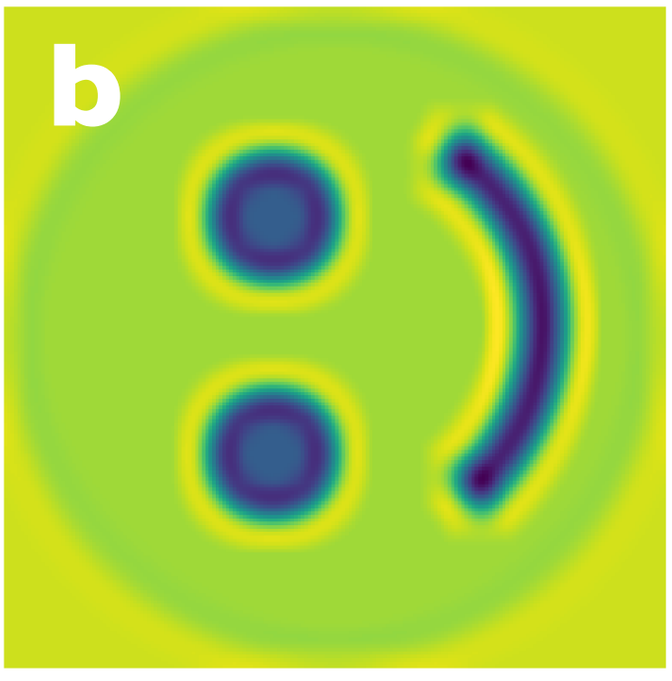
What was the kernel K?

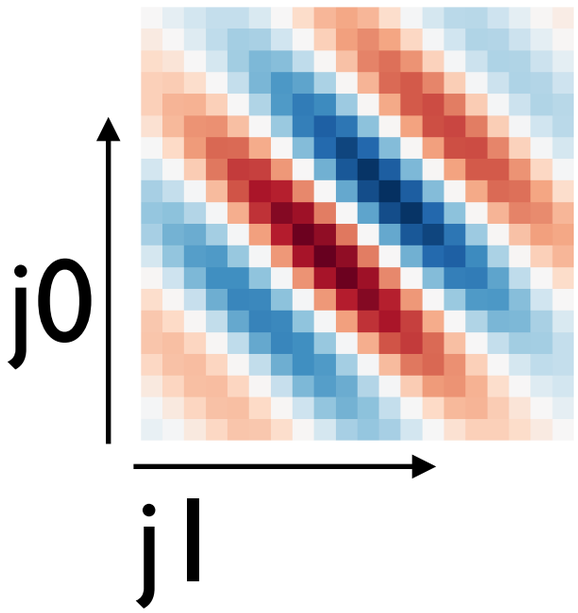
a [-2,0,2] **c** [-1,0,1] **e** [1,2,1]

b [-1,2,-1] **d** [1,-2,1] **f** [1,0,-1]

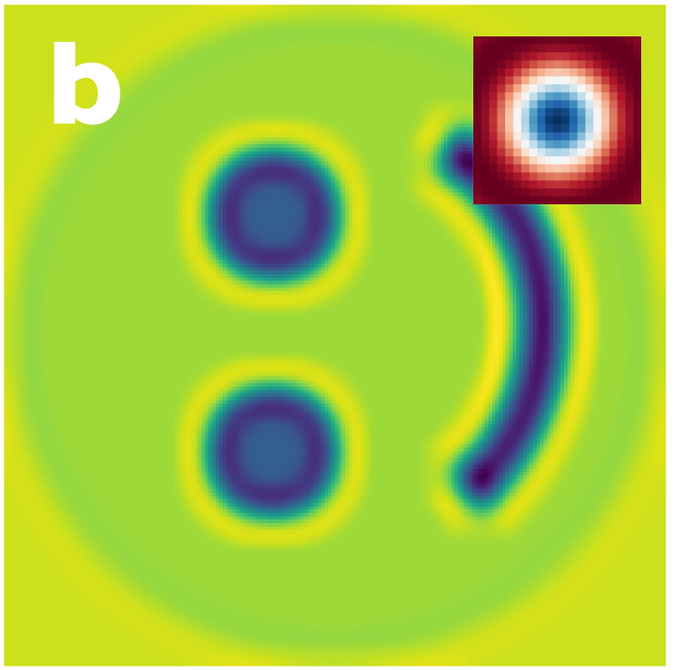


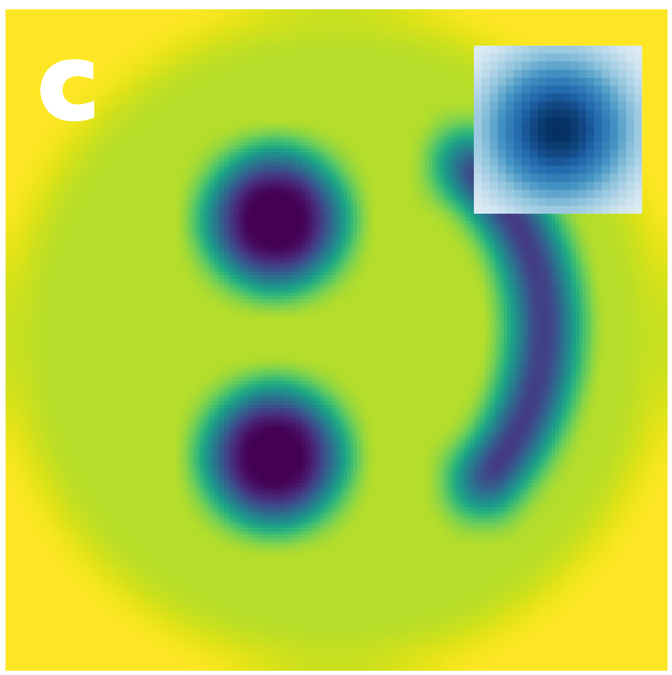
Kernel K





Kernel K





```
net.add(Conv2D(input_shape=(N,N,1),
filters=20, kernel_size=[3,3],
activation='relu',padding='same'))

net.add(Conv2D(
filters=5, kernel_size=[4,4],
activation='relu',padding='same'))
```

What is the total amount of weight values stored for the transition between these two layers?

a 5*20*4*3

b 5*4*4

c 5*5*4*4

d 5*20*4*4*3*3

e 5*20*4*4

f N*N*20*5*4*3

```
net.add(Conv2D(input_shape=(N,N,1),
filters=20, kernel_size=[3,3],
activation='relu',padding='same'))

net.add(Conv2D(
filters=5, kernel_size=[4,4],
activation='relu',padding='same'))
```

What is the total amount of weight values stored for the transition between these two layers?

a 5*20*4*3

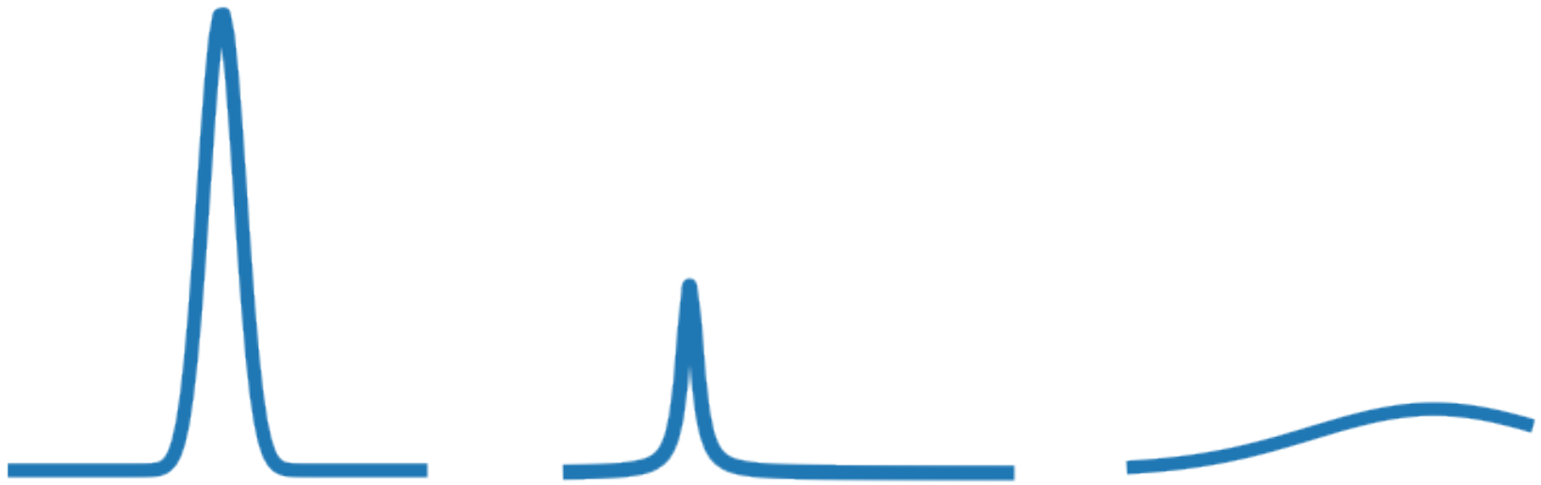
b 5*4*4

c 5*5*4*4

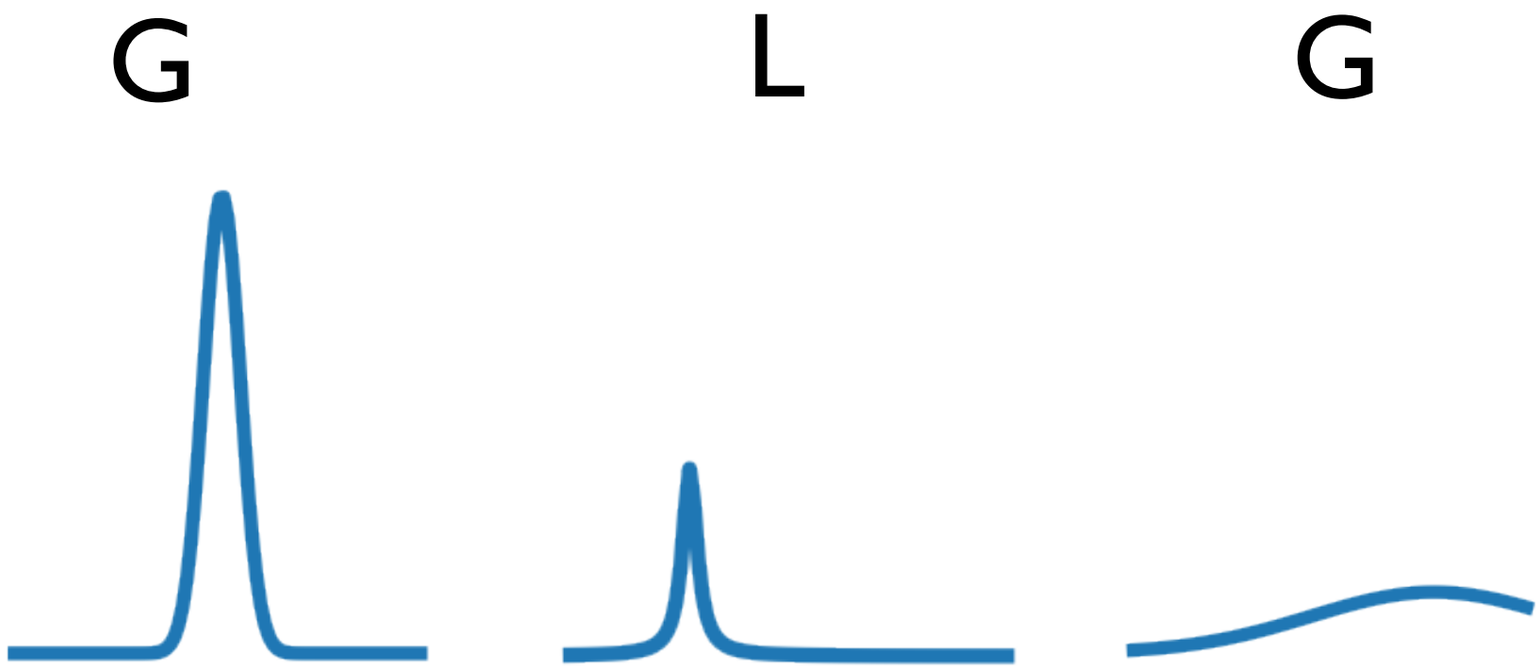
d 5*20*4*4*3*3

e 5*20*4*4

f N*N*20*5*4*3



Gaussian (G) or Lorentzian (L) ?



Gaussian (G) or Lorentzian (L) ?

Lecture 4 Homework

(will be briefly discussed in the online session for lecture 5)

Machine Learning for Physicists

1. * Compete in the Grand Autoencoder Challenge!

2. Train a network to turn multiple random circles into squares of the same size and placement!

Machine Learning for Physicists

Grand Autoencoder Challenge

Medium Challenge (9 bottleneck neurons)

normal training (30,000 images seen)

Dimitry	0,0039
Francesco	0,005
Pablo	0,0054
Luca F	0,0058
Martin	0,00595
Lukas	0,0061

long training (100,000 images)

Khabab 0.0022, Mangesh 0.0024, Francesco 0.0029

Hard Challenge (3 bottleneck neurons)
normal training (30,000 images seen)

Martin ?0.0011 "with large fluctuations"
Dimitriy 0.0050
Lukas 0.008
Francesco 0.0082

long training (100,000 images)

Lukas 0.0048
Francesco 0.0049
Conrad 0.0074
Jalil 0.011

Dimitriy Ayzenberg's network

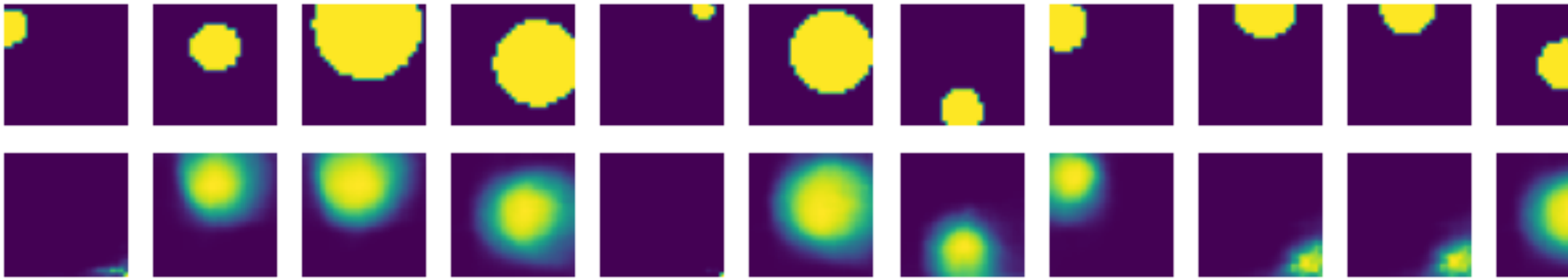
```
Net.add(Conv2D(27,9,input_shape=(27,27,1),
                activation="tanh",padding='same', kernel_initializer='random_uniform',
                     bias_initializer='zeros'))
  Net.add(AveragePooling2D(pool_size=3,padding='same')) # down
  Net.add(Conv2D(27,9,
                activation="relu",padding='same', kernel_initializer='random_uniform',
                     bias_initializer='zeros'))
  Net.add(AveragePooling2D(pool_size=3,padding='same')) # down
  Net.add(Conv2D(3,9,
                activation="relu",padding='same', kernel_initializer='random_uniform',
                     bias_initializer='zeros'))
  Net.add(AveragePooling2D(pool_size=3,padding='same')) # down
  Net.add(UpSampling2D(size=3)) # up
  Net.add(Conv2DTranspose(27,9,
                activation="relu",padding='same', kernel_initializer='random_uniform',
                     bias_initializer='zeros'))
  Net.add(UpSampling2D(size=3)) # up
  Net.add(Conv2DTranspose(27,9,
                activation="relu",padding='same', kernel_initializer='random_uniform',
                     bias_initializer='zeros'))
  Net.add(UpSampling2D(size=3)) # up
  Net.add(Conv2DTranspose(27,9,
                  activation="relu",padding='same', kernel_initializer='random_uniform',
                     bias_initializer='zeros'))
  Net.add(UpSampling2D(size=3)) # up
  Net.add(Conv2DTranspose(27,9,
                  activation="relu",padding='same', kernel_initializer='random_uniform',
                     bias_initializer='zeros'))
  Net.add(UpSampling2D(size=3)) # up
  Net.add(Conv2DTranspose(27,9,
                  activation="tanh",padding='same', kernel_initializer='random_uniform',
                     bias_initializer='zeros'))
  Net.add(AveragePooling2D(pool_size=9,padding='same')) # down
  Net.compile(loss='mean_squared_error',
                optimizer=tf.keras.optimizers.Adam(learning_rate=tf.keras.optimizers.schedule
s.PolynomialDecay(initial_learning_rate=0.001, decay_steps=1000, end_learning_rate=0.00001)))
```

```
Layer 0: 19683 neurons /  (30, 27, 27, 27)
Layer 1: 2187 neurons /  (30, 9, 9, 27)
Layer 2: 2187 neurons /  (30, 9, 9, 27)
Layer 3: 243 neurons /  (30, 3, 3, 27)
Layer 4: 27 neurons /  (30, 3, 3, 3)
Layer 5: 3 neurons /  (30, 1, 1, 3)
Layer 6: 27 neurons /  (30, 3, 3, 3)
Layer 7: 243 neurons /  (30, 3, 3, 27)
Layer 8: 2187 neurons /  (30, 9, 9, 27)
Layer 9: 2187 neurons /  (30, 9, 9, 27)
Layer 10: 19683 neurons /  (30, 27, 27, 27)
Layer 11: 19683 neurons /  (30, 27, 27, 27)
Layer 12: 177147 neurons /  (30, 81, 81, 27)
Layer 13: 177147 neurons /  (30, 81, 81, 27)
Layer 14: 1594323 neurons /  (30, 243, 243, 27)
Layer 15: 1594323 neurons /  (30, 243, 243, 27)
Layer 16: 19683 neurons /  (30, 27, 27, 27)
                                          ?
```

"Hardest challenge" (1 neuron)

Long training (100000 images) cost: 0.069 +/- 0.002

Francesco Graffitti



HARDEST (1 neuron)

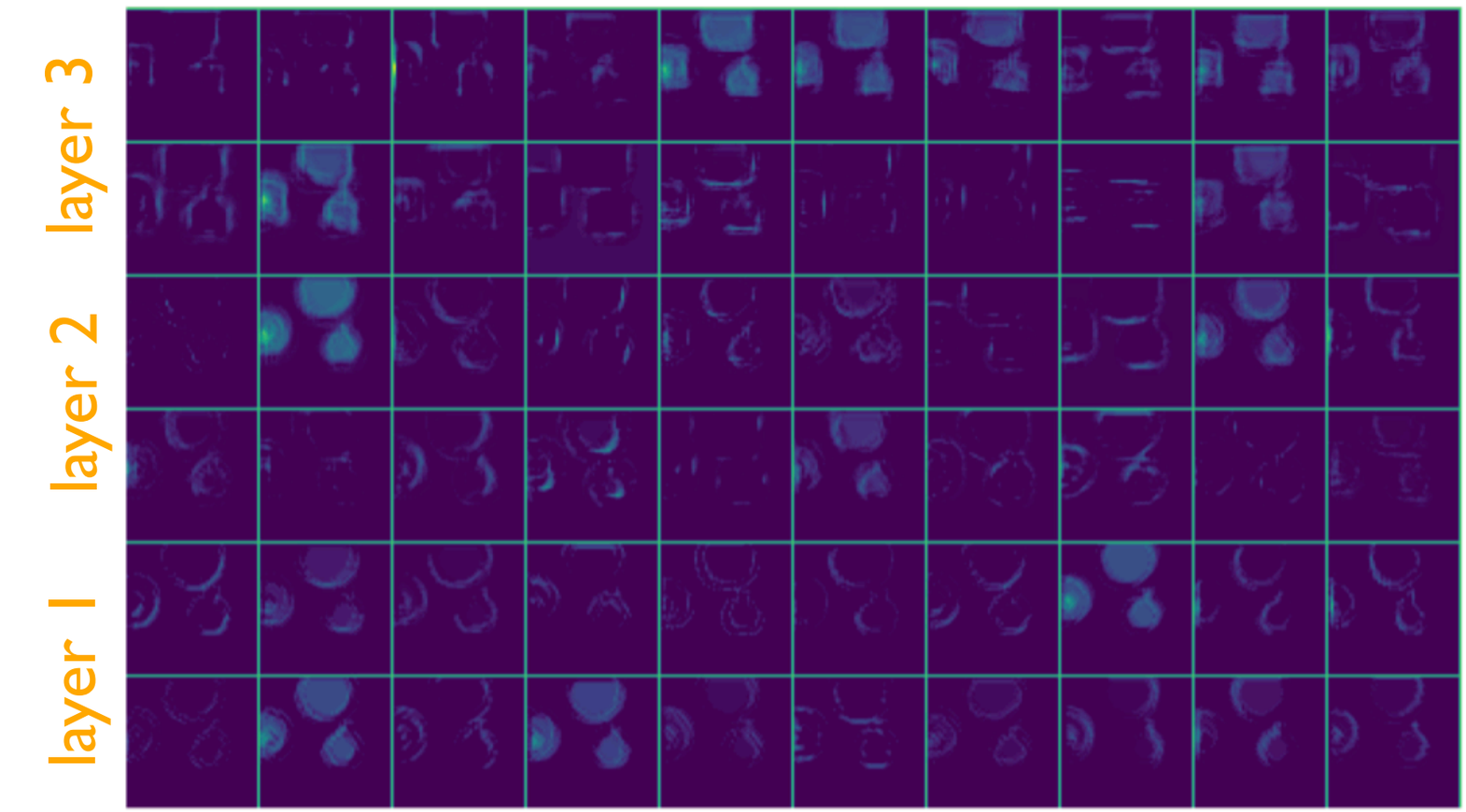
```
lrelu = lambda x: relu(x, alpha=0.1)
sigmoidMod = lambda x: sigmoid(5*x)

Net.add(Input(shape=(27,27,1)))
Net.add(Conv2D(32,3,activation=lrelu,padding='same'))
Net.add(MaxPooling2D(pool_size=(3,3),padding='same')) # down
Net.add(Conv2D(64,3,activation=lrelu,padding='same'))
Net.add(MaxPooling2D(pool_size=(3,3),padding='same')) # down
Net.add(Conv2D(128,3,activation=lrelu,padding='same'))
Net.add(MaxPooling2D(pool_size=(2,2),padding='same')) # down

Net.add(Flatten())
Net.add(Dense(1,activation='linear'))
Net.add(Dense(576,activation=lrelu))
Net.add(Reshape((3,3,64)))

Net.add(Conv2D(128,3,activation=lrelu,padding='same'))
Net.add(UpSampling2D(size=(3,3)))
Net.add(Conv2D(64,3,activation=lrelu,padding='same'))
Net.add(UpSampling2D(size=(3,3)))
Net.add(Conv2D(32,3,activation=lrelu,padding='same'))
Net.add(Conv2D(1,3,activation=sigmoidMod, padding='same'))

Net.compile(loss='mean_squared_error',optimizer=Adam(learning_rate=0.0005))
```

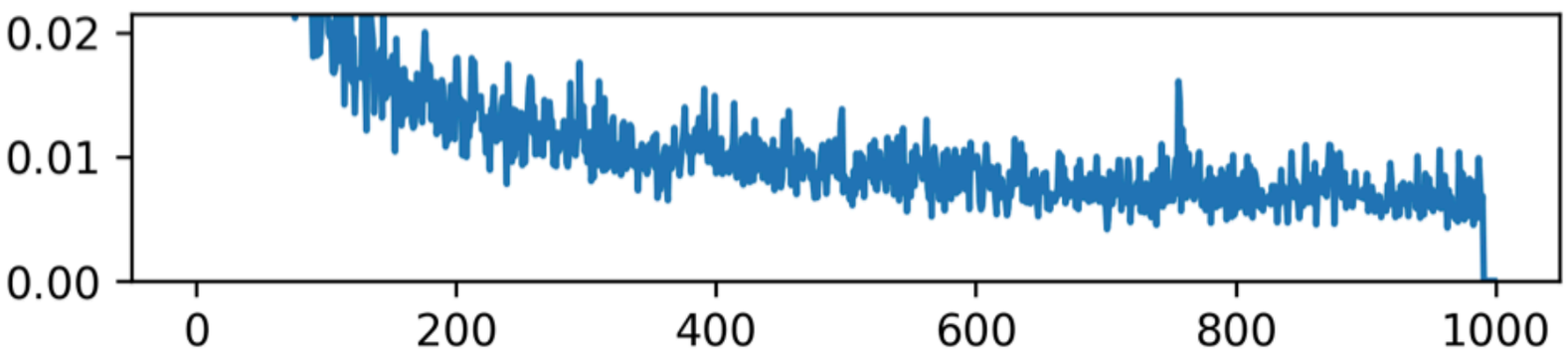
Special mention by the jury

Turning randomly placed circles into squares

(50x50 images with 5 randomly placed circles)

4 hidden layers, 20 channels, kernel size 5x5, "relu"
output layer has 1 channel, and "linear"



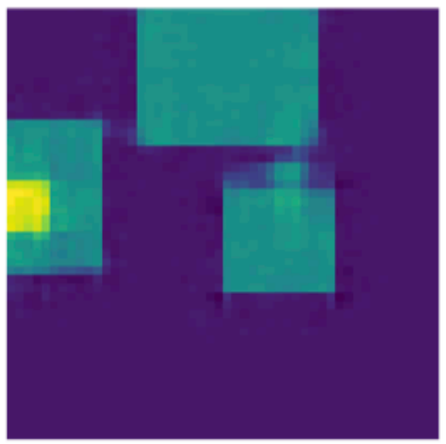


(batch size 10: 10k training images seen)

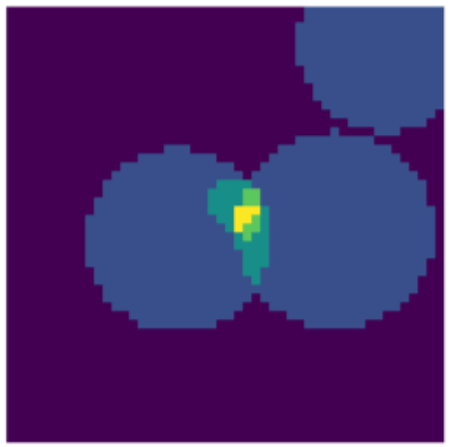
test input

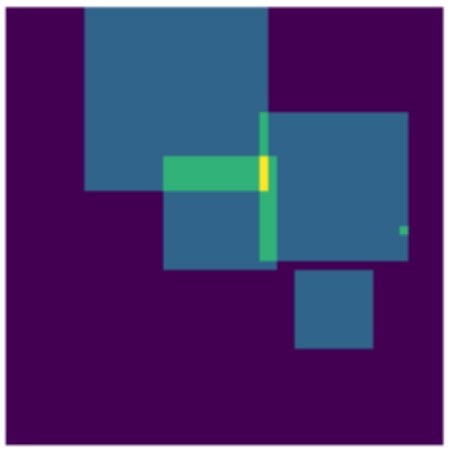
test target

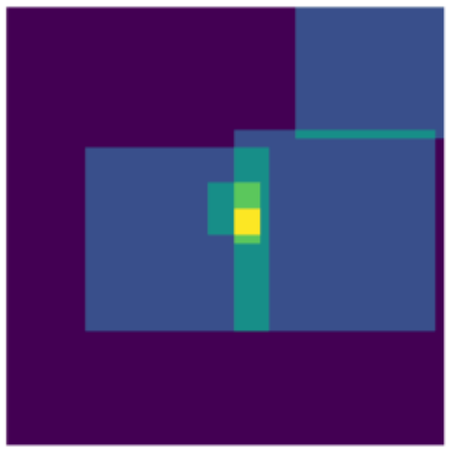
test NN output



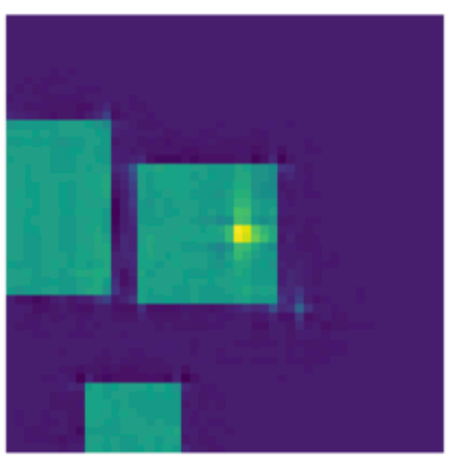
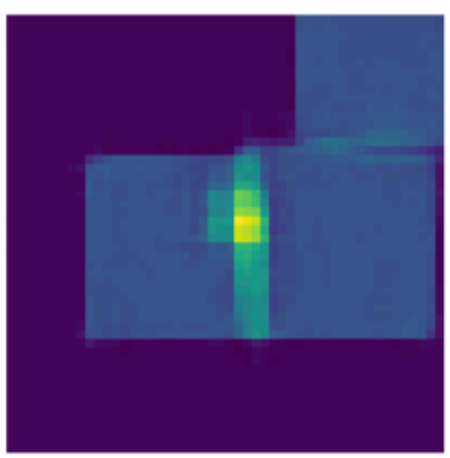
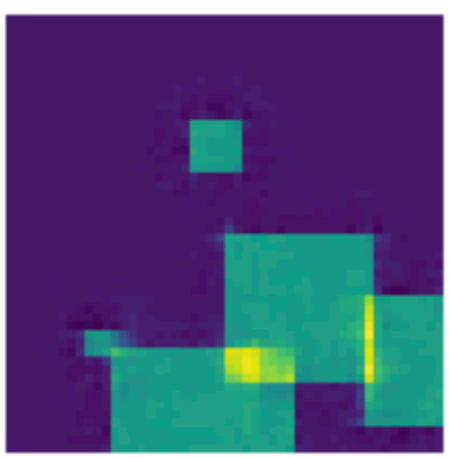




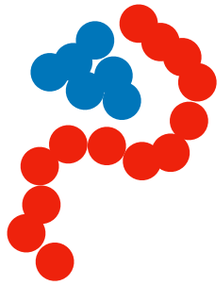




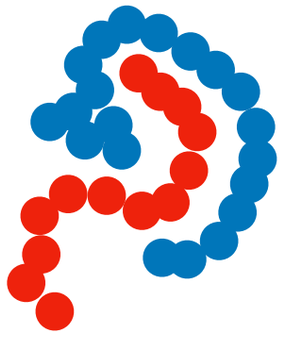





Which of the following clusters would be hardest to disentangle correctly for a technique like t-SNE?

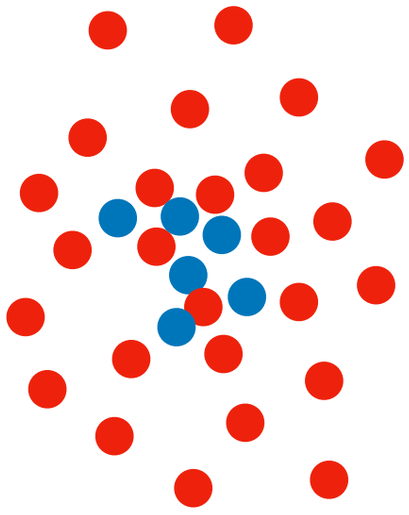
a



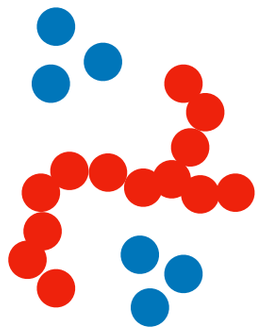
b



c

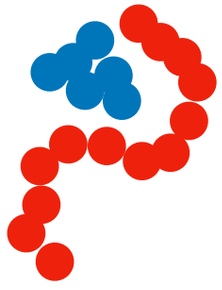


d

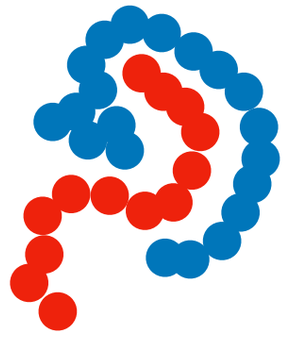


Which of the following clusters would be hardest to disentangle correctly for a technique like t-SNE?

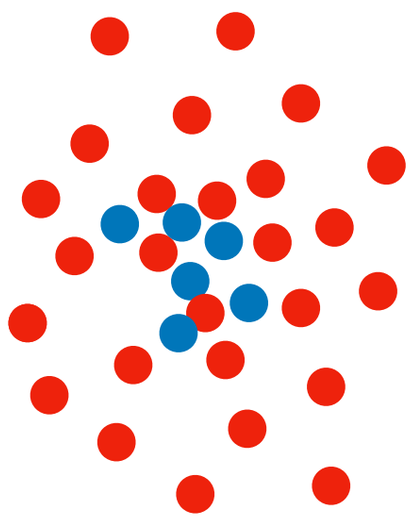
a



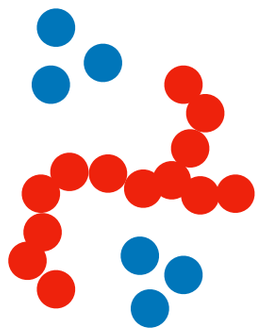
b



c



d



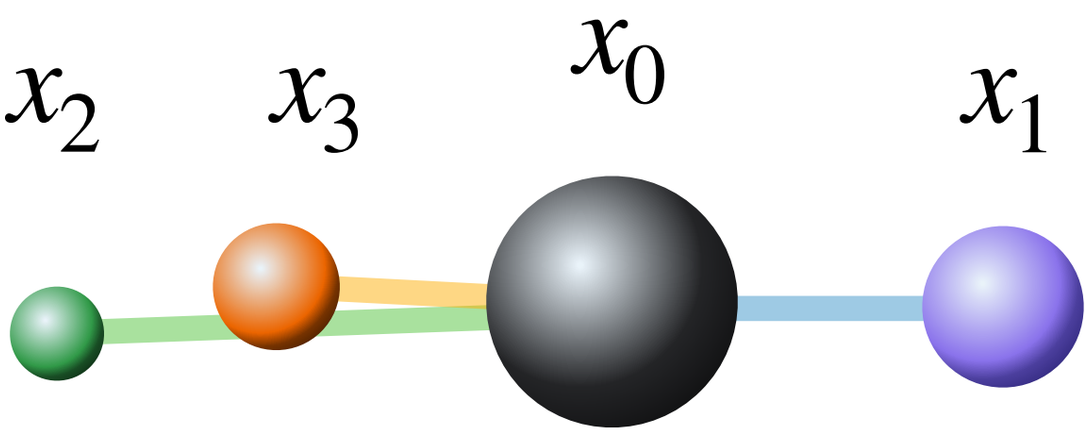
Lecture 5 Homework

(will be briefly discussed in the online session for lecture 6)

Machine Learning for Physicists

* I Visualize by t-SNE the inner layer activations of a trained autoencoder (trained on two different kinds of images)

Tutorial: predicting physical time evolution



Lecture 6 Homework

(will be briefly discussed in the online session for lecture 7)

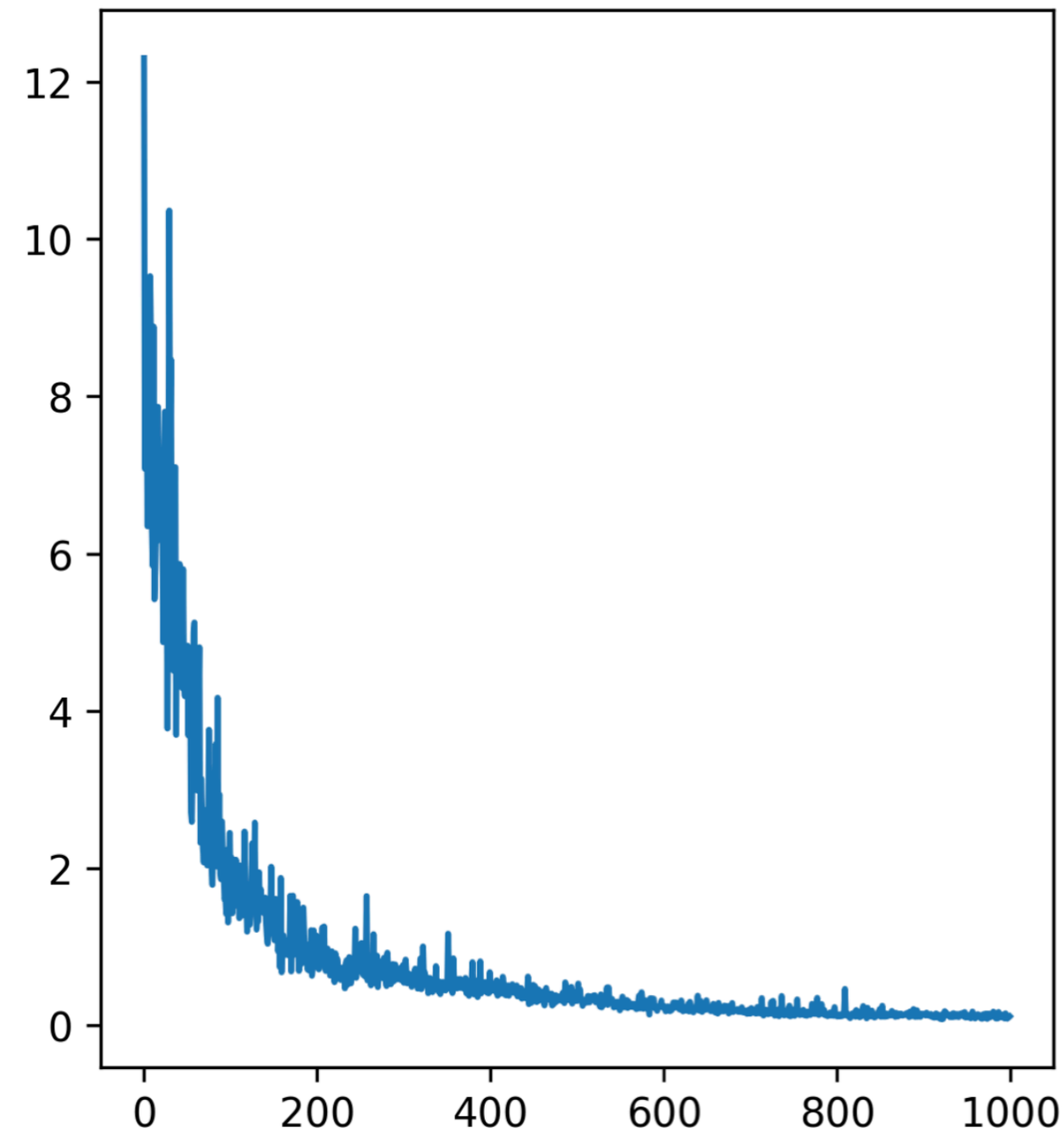
Machine Learning for Physicists

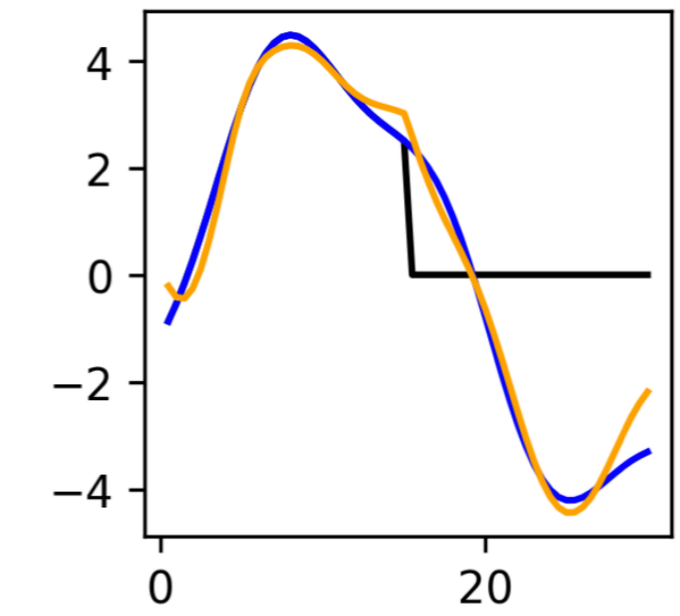
* I

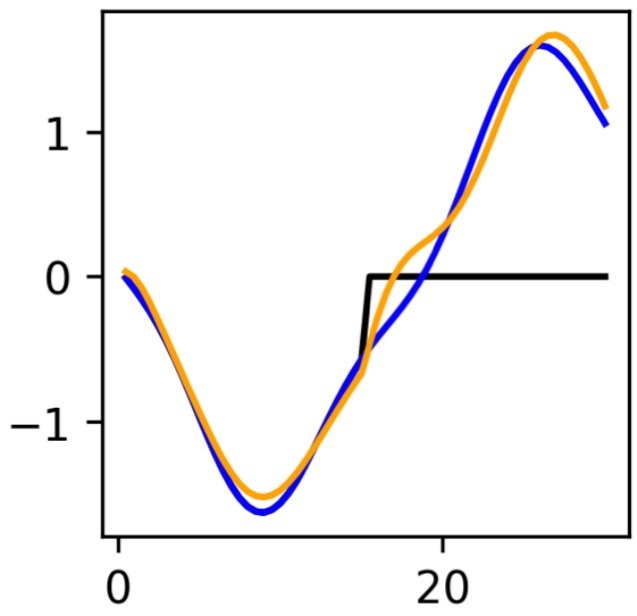
Apply a recurrent network to predict the time-evolution of a physical system during t=T..2T after observing it during t=0..T (for random initial conditions); try to treat the case of 'partial' observation (i.e. observing only one of the particles)

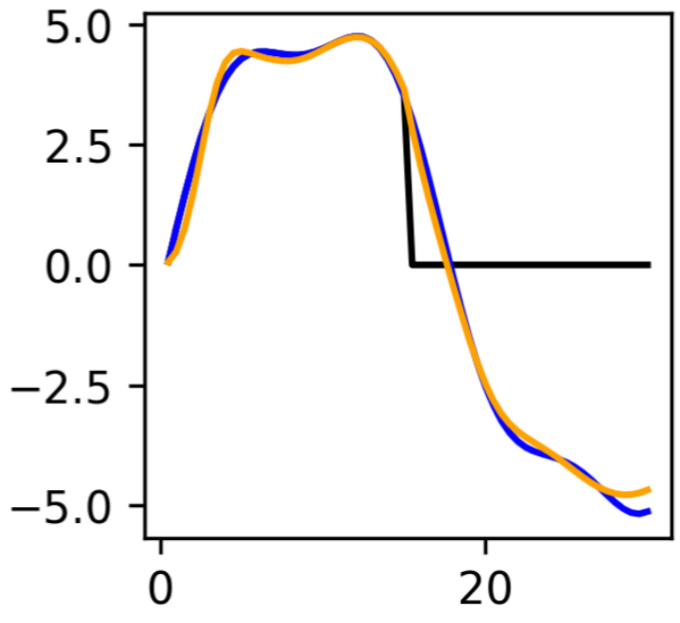
2 particles, observe particle '0' during [0,T], predict its motion for [0,2*T]

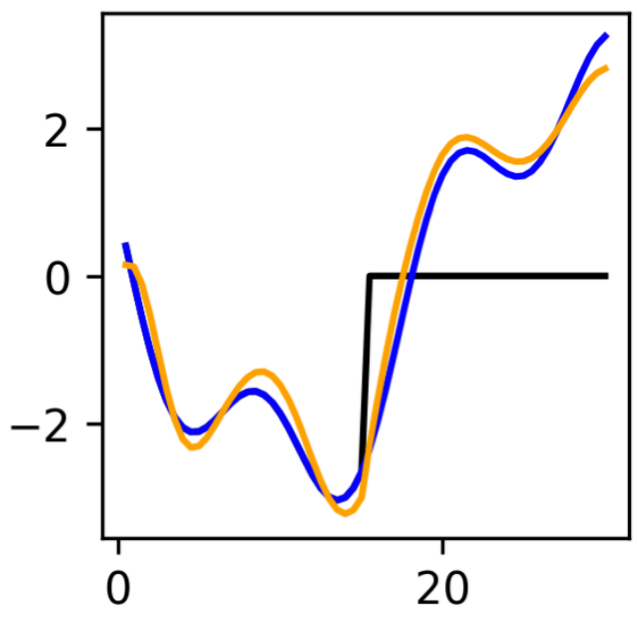
```
the_input=np.copy(Xs[0,0,:,:][:,:,None])
the_input[:,int(nsteps/2):,0]=0.0 # delete second half!
```







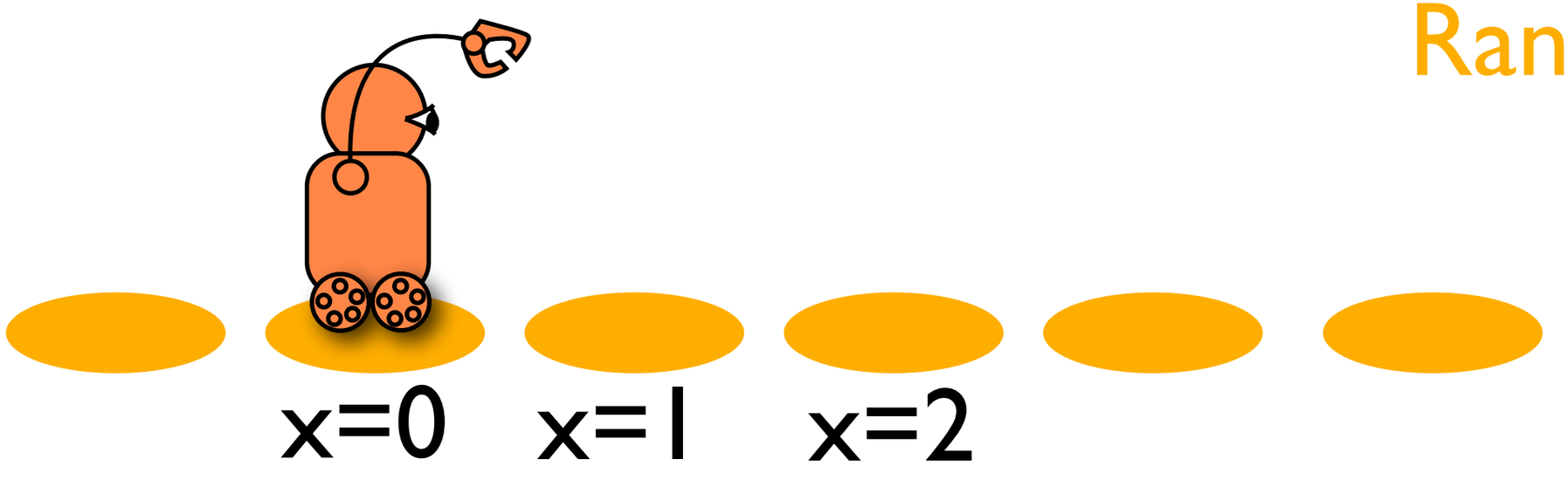




Quiz

Machine Learning for Physicists

Random walk



10 steps, left/right with 50% probability each

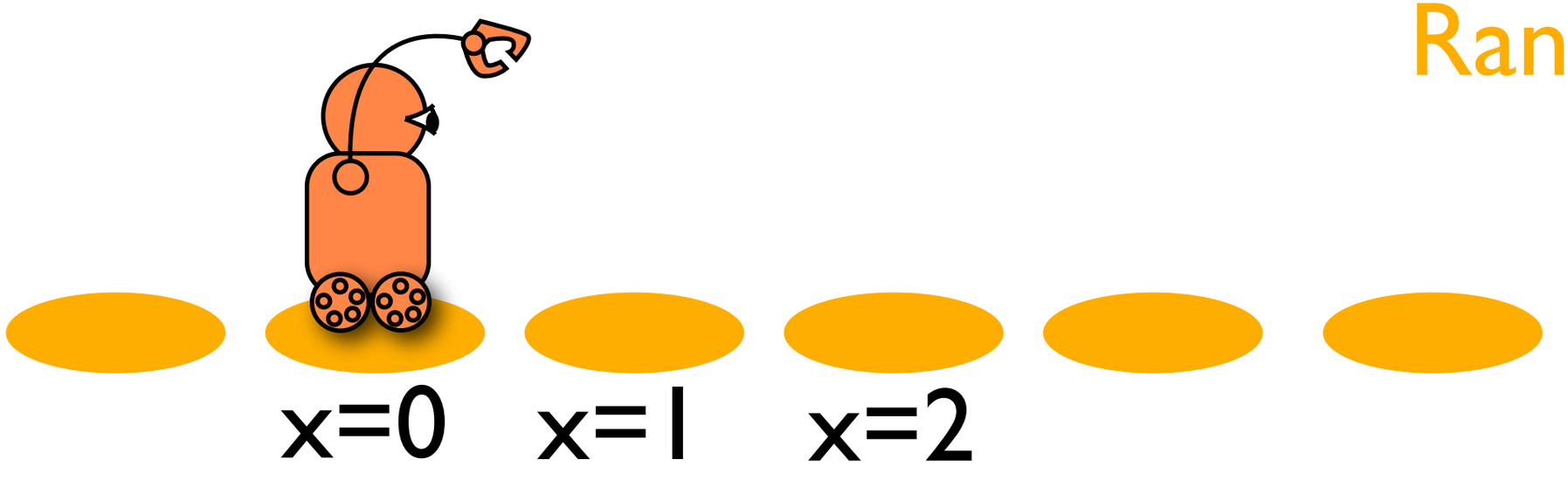
R=1 only if x=2 at the end

What is the probability for this to happen?

a $\binom{10}{2}\left(\frac{1}{2}\right)^{10}$ **b** $\binom{10}{2}\left(\frac{1}{2}\right)^{2}$ **c** $\left(\frac{1}{2}\right)^{10}$

d $\binom{10}{4}\left(\frac{1}{2}\right)^{10}$ **e** $\binom{10}{4}\left(\frac{1}{2}\right)^{4}$

Random walk



10 steps, left/right with 50% probability each

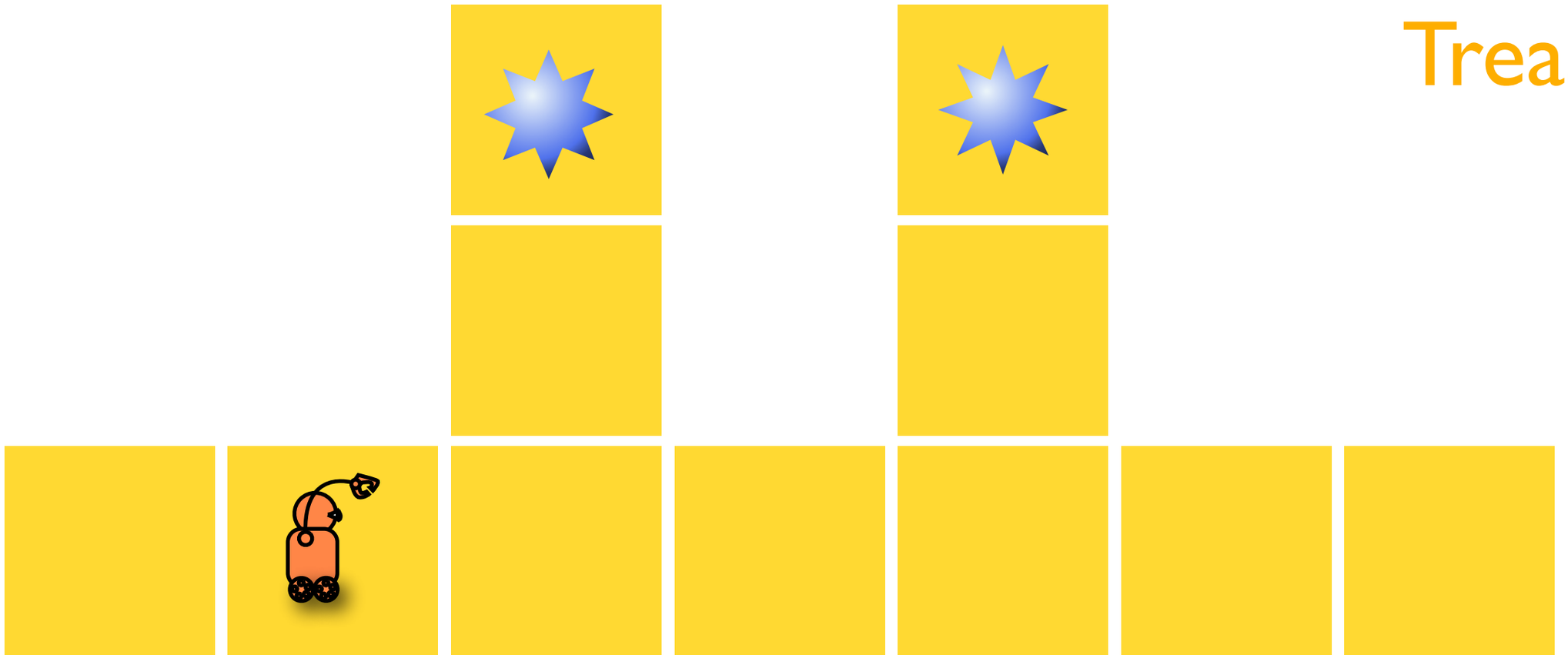
R=1 only if x=2 at the end

What is the probability for this to happen?

a $\binom{10}{2}\left(\frac{1}{2}\right)^{10}$ **b** $\binom{10}{2}\left(\frac{1}{2}\right)^{2}$ **c** $\left(\frac{1}{2}\right)^{10}$

d $\binom{10}{4}\left(\frac{1}{2}\right)^{10}$ **e** $\binom{10}{4}\left(\frac{1}{2}\right)^{4}$

Treasures



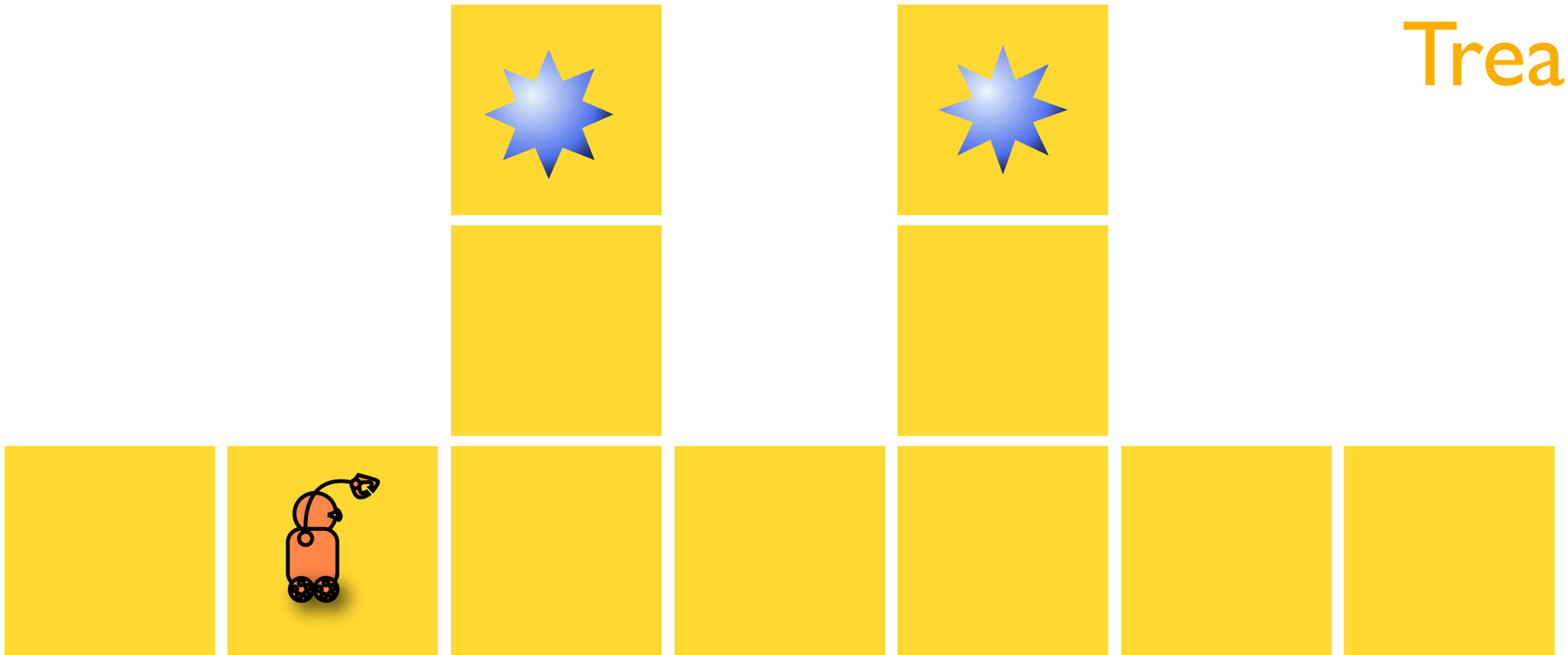
each step: go in indicated direction (if possible, else stay)
each step: get reward +1 if on diamond site
10 steps. The robot gets the best return for…

a → ↑ ↑ → → → → → → →

b → ↑ ↑ ↓ ↓ → → ↑ ↑ ↑

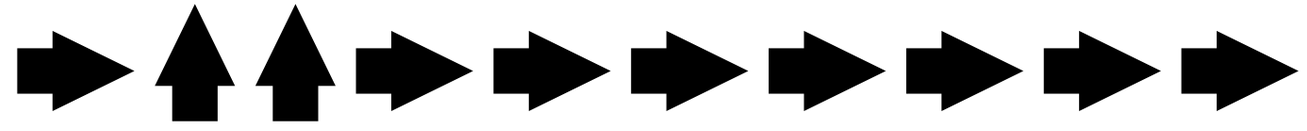
c → → → ↑ ↑ → → → → →

Treasures

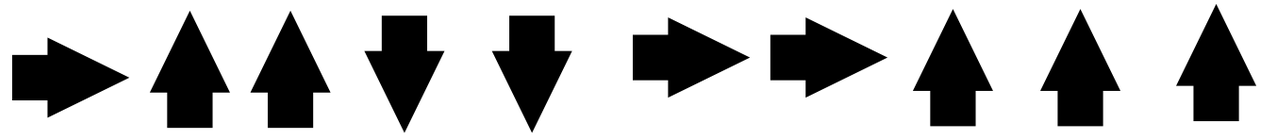


each step: go in indicated direction (if possible, else stay)
each step: get reward +1 if on diamond site
10 steps. The robot gets the best return for…

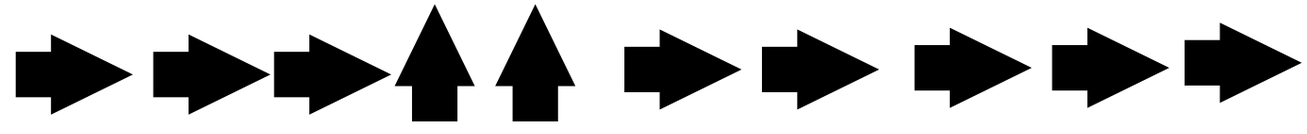
a



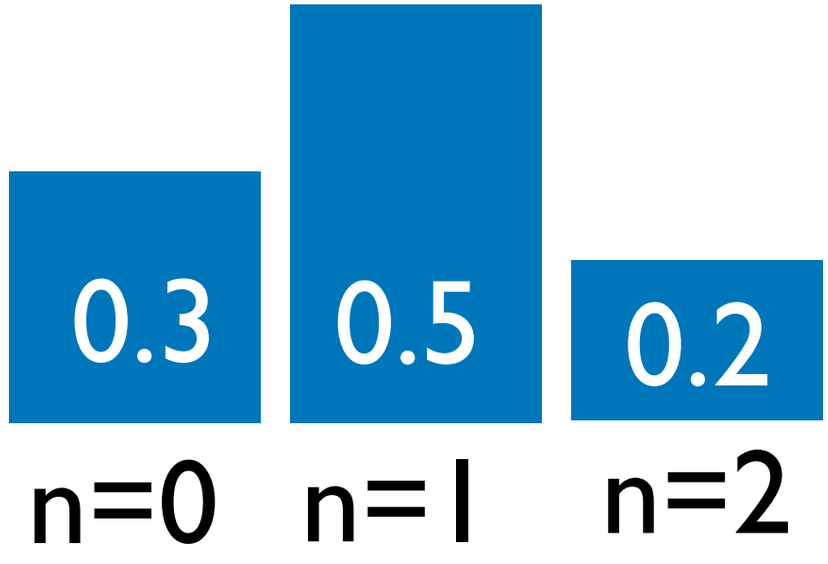
b



c



Arbitrary probabilities



How to return a random n distributed according to this probability distribution?

```
p=np.random.uniform(low=0,high=1)
```

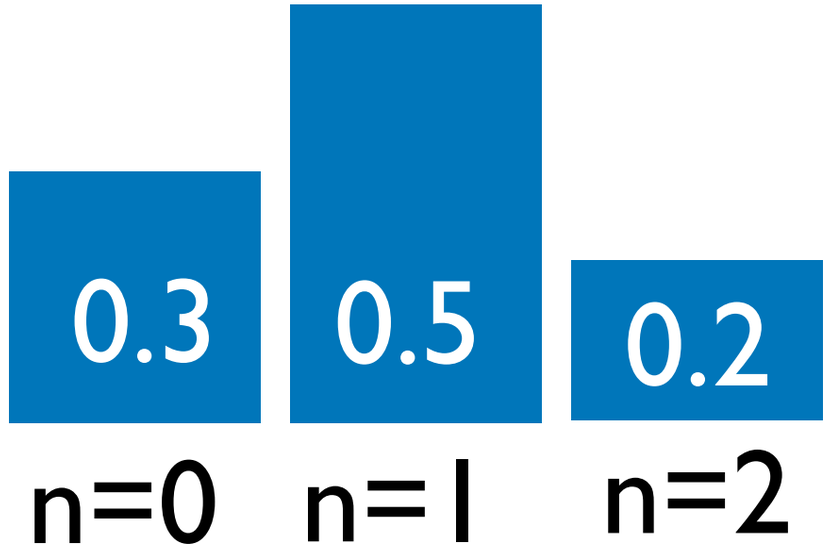
a
```
if p<0.3: return(0)
if p<0.5: return(1)
if p<0.2: return(2)
```

b
```
if p>0.3: return(0)
if p>0.5: return(1)
if p>0.2: return(2)
```

c
```
if p<0.3: return(0)
if p<0.8: return(1)
if p<1: return(2)
```

d
```
if p>0.3: return(0)
if p>0.8: return(1)
if p>1: return(2)
```

Arbitrary probabilities



How to return a random n distributed according to this probability distribution?

```
p=np.random.uniform(low=0,high=1)
```

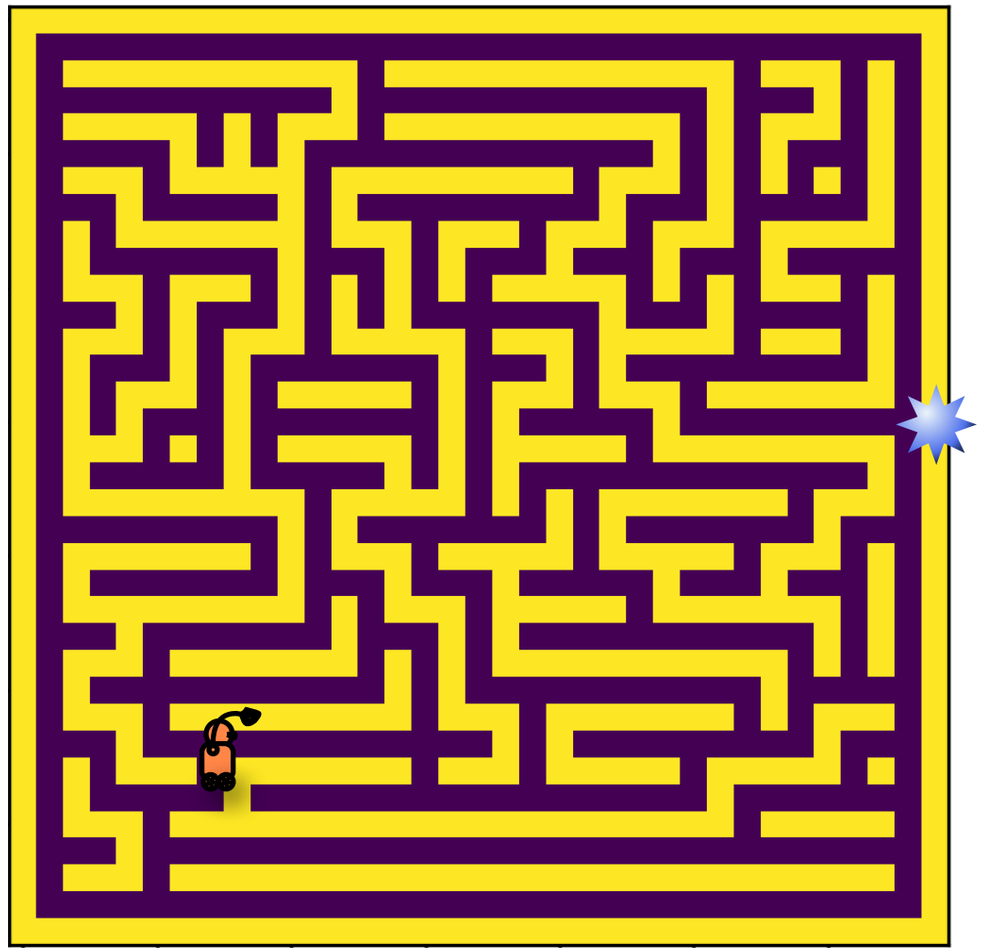
a
```
if p<0.3: return(0)
if p<0.5: return(1)
if p<0.2: return(2)
```

b
```
if p>0.3: return(0)
if p>0.5: return(1)
if p>0.2: return(2)
```

c
```
if p<0.3: return(0)
if p<0.8: return(1)
if p<1: return(2)
```

d
```
if p>0.3: return(0)
if p>0.8: return(1)
if p>1: return(2)
```

Maze



35x35 maze

We want the robot to find the target in an arbitrary maze, so we provide the full maze image as the state!

What is the shape of policy[state,action] as an array?

a `[35,35,4]`

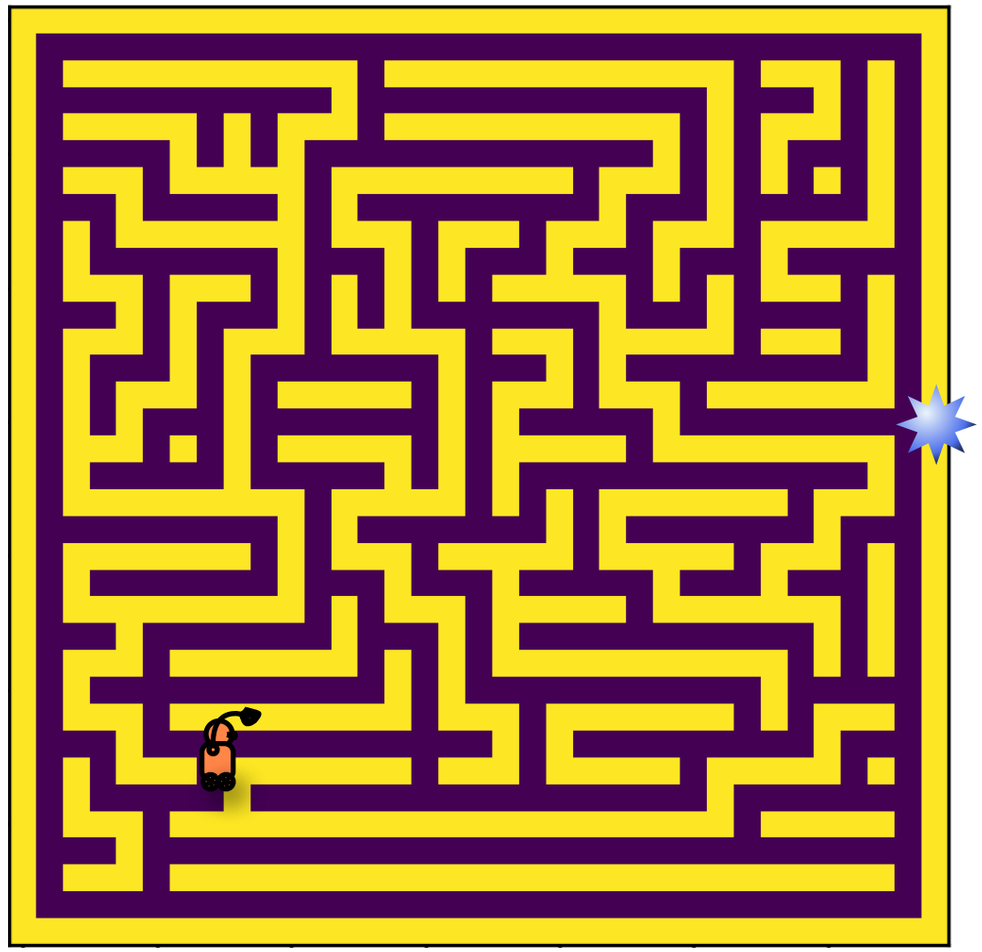
b `[35**2,4]`

c `[2**35,4]`

d `[2**(35**2),4]`

e `[2**35,2**35,4]`

Maze



35x35 maze

We want the robot to find the target in an arbitrary maze, so we provide the full maze image as the state!

What is the shape of policy[state,action] as an array?

a `[35,35,4]`

b `[35**2,4]`

c `[2**35,4]`

d `[2**(35**2),4]`

e `[2**35,2**35,4]`

Lecture 7 Homework

(will be briefly discussed in the online session for lecture 8)

Machine Learning for Physicists

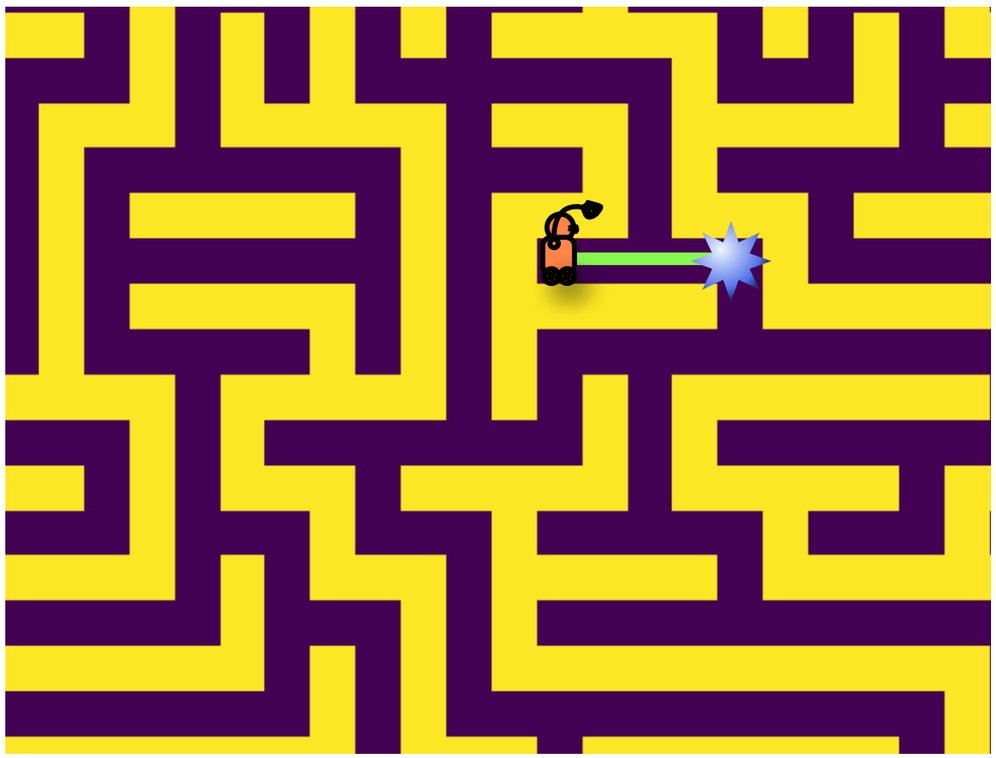
*

I

Invent a modified maze-game for the robot, with different rules, such that the state space needs to be enlarged!

'look-ahead': Additional information in state that says whether a treasure chest is to the north/south/west/east (on next site or with free path); will multiply(!) state space by factor 5

policy[jx,jy,direction] becomes
policy[jx,jy,look-ahead,direction]



'see-surroundings': robot can see 8 surrounding sites

'plan-ahead': robot has device that checks whether any of the paths EN, ES, NE, etc. would lead to a treasure chest; where EN means: walk to the east until there is an opening to the north, then walk to the north until you hit a chest…

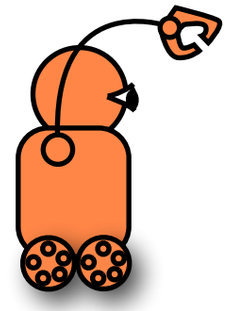
with ghosts: like in PacMan, ghosts run randomly through the maze, will give negative reward – robot has to avoid them (can extend any of these strategies above…)

Quiz

Machine Learning for Physicists

'see-surroundings': robot can see 8 surrounding sites

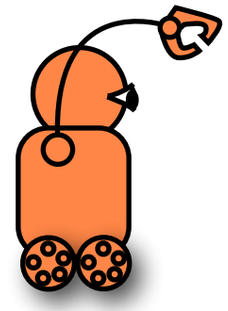
By which factor does this enlarge the state space?



a 8 **b** 2^8 **c** 3^8 **d** 4^8

'see-surroundings': robot can see 8 surrounding sites

By which factor does this enlarge the state space?



a 8 **b** 2^8 **c** 3^8 **d** 4^8

Which is the smallest convolutional network that could produce a policy for the robot that is influenced by the position of the chest?

channels / kernel_size

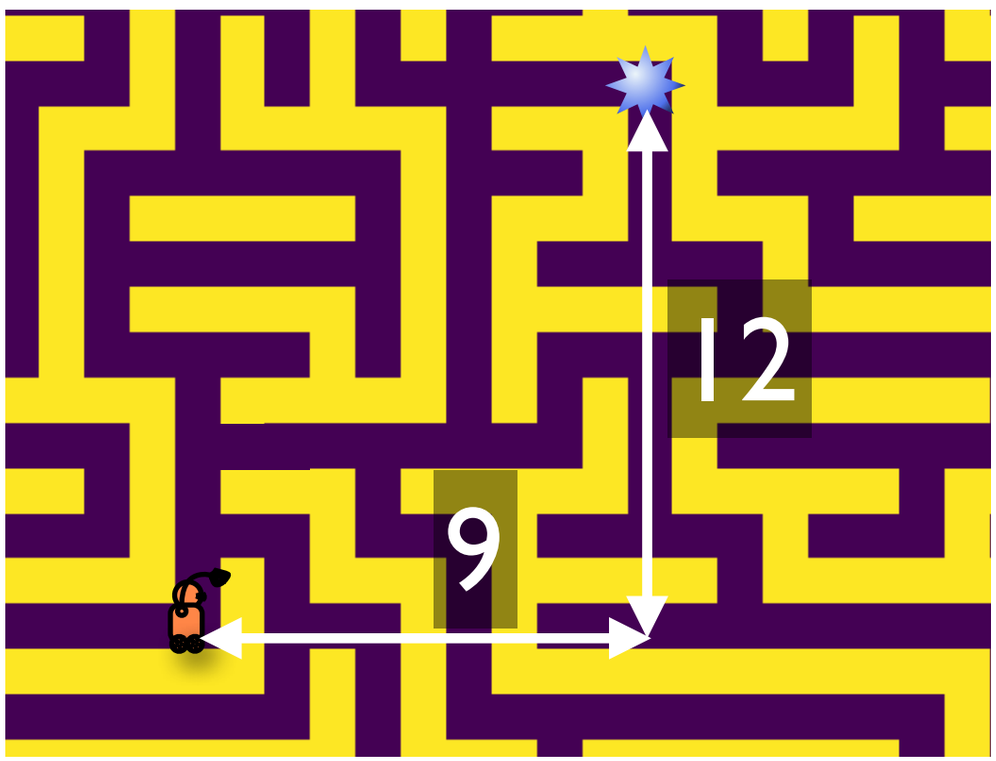
a 8 3
20 5
10 3

b 8 5
10 7
10 7
5 5
5 5

c 8 5
10 7
10 5

d 8 5
10 3
10 7
5 5
5 3

(after the last conv. layer: just pooling by summation over all pixels)



Which is the smallest convolutional network that could produce a policy for the robot that is influenced by the position of the chest?

channels / kernel_size

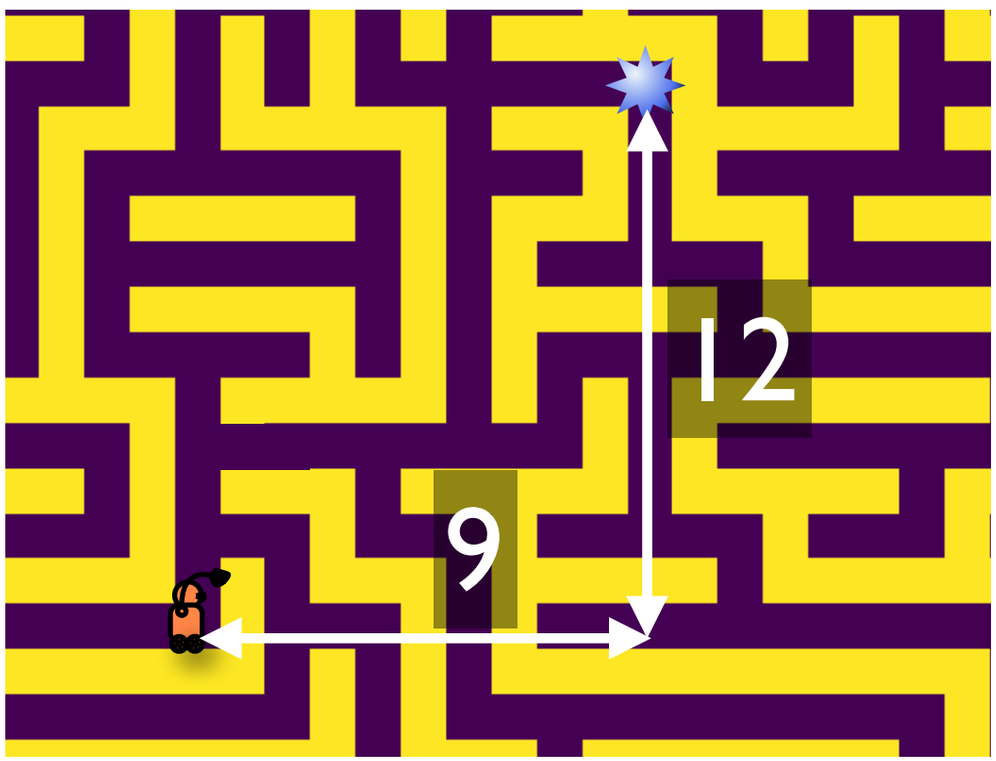
a 8 3
20 5
10 3

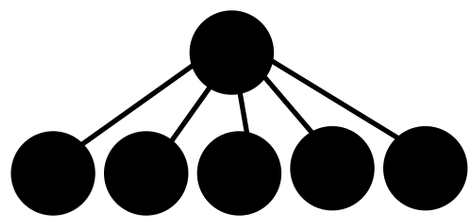
b 8 5
10 7
10 7
5 5
5 5

c 8 5
10 7
10 5

d 8 5
10 3
10 7
5 5
5 3

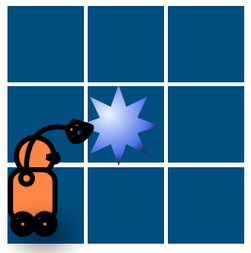
(after the last conv. layer: just pooling by summation over all pixels)





kernel_size=5

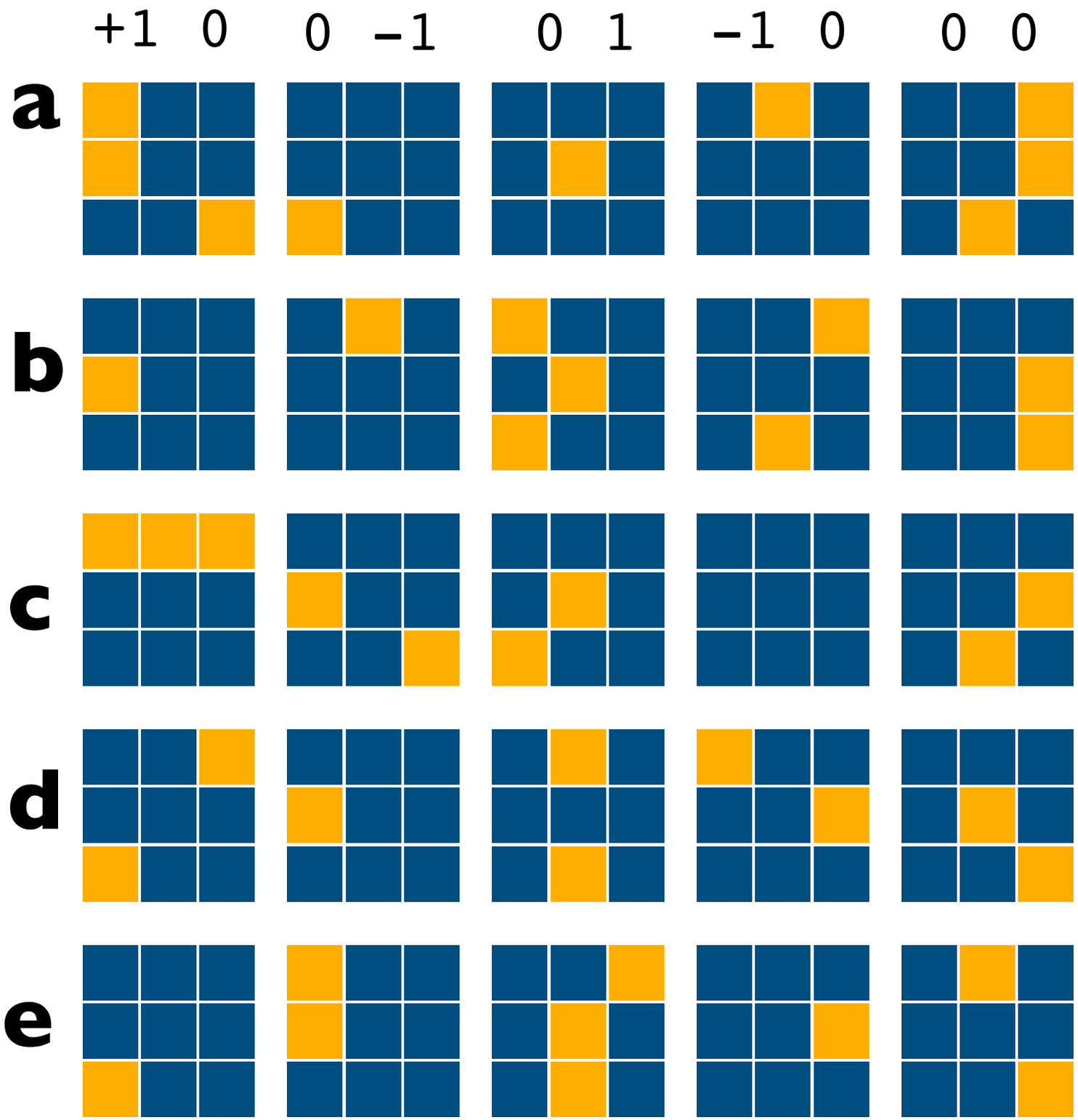
Which is the best policy for the robot?
(treasure does not get deleted, 10 time steps)



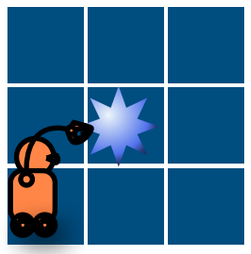
0

1

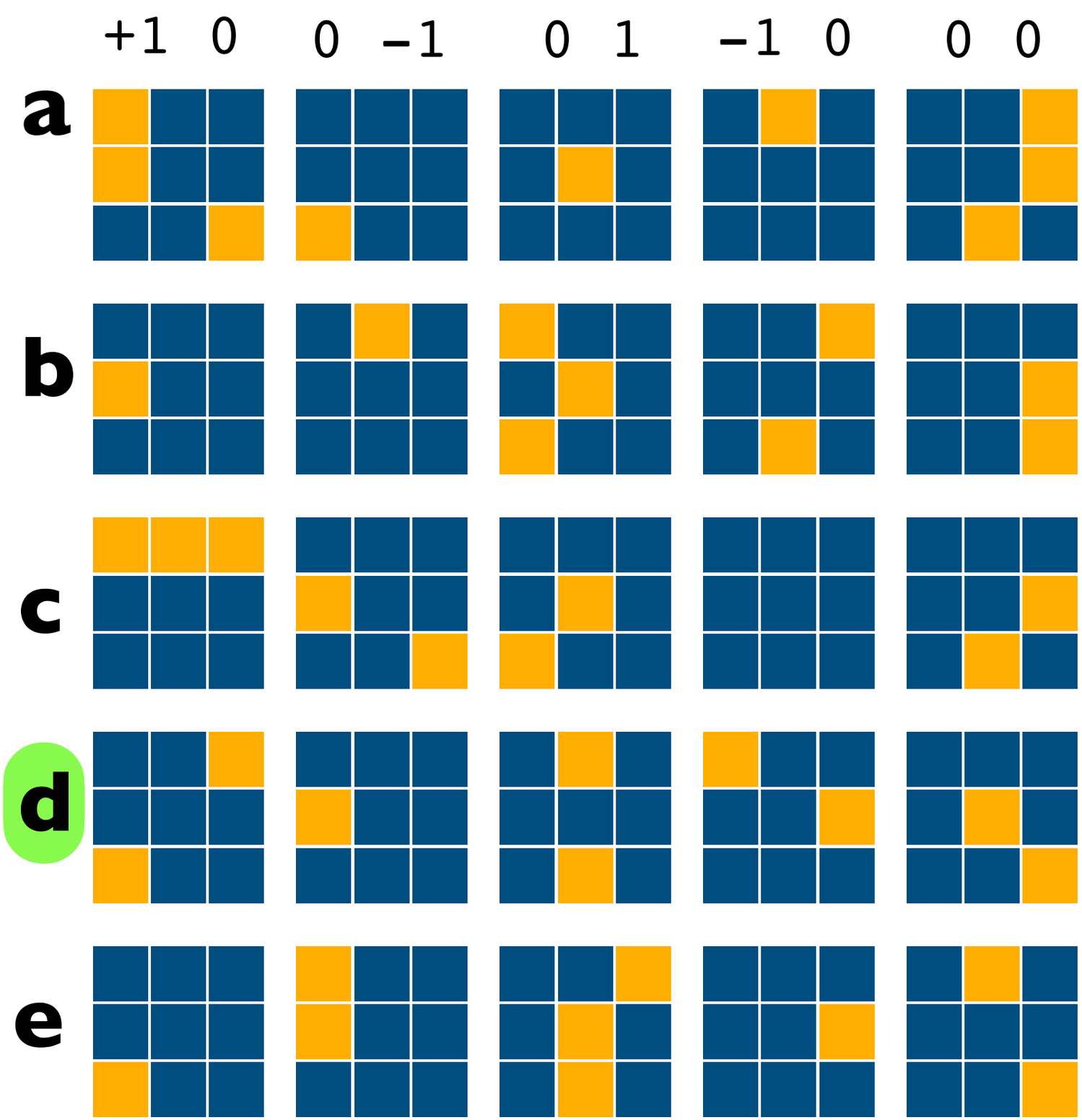
Each panel shows the policy for that action (dx,dy) on the grid



Which is the best policy for the robot?
(treasure does not get deleted, 10 time steps)



Each panel shows the policy for that action (dx,dy) on the grid



Lecture 8 Homework

(will be briefly discussed in the online session for lecture 9)

Machine Learning for Physicists

* Try to optimize the network-based policy gradient (eventually: for mazes instead of an empty grid!)

"It doesn't work" – what to do?

Check in very simplest test cases whether there is still some bug in the program

Improve visualization for better testing

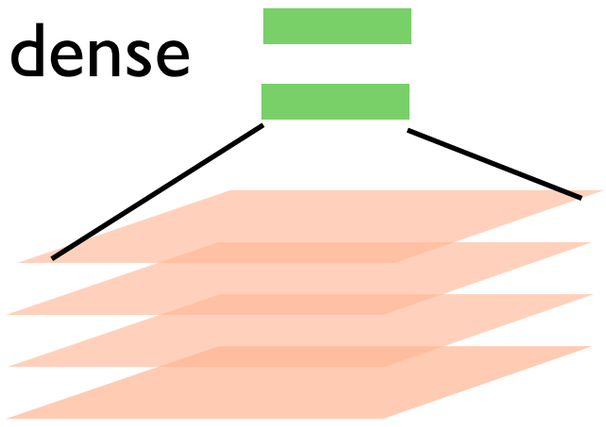
Check influence of parameters, in cases where this is feasible without waiting too long

Improve ease-of-use of the program

Change concept

Old concept

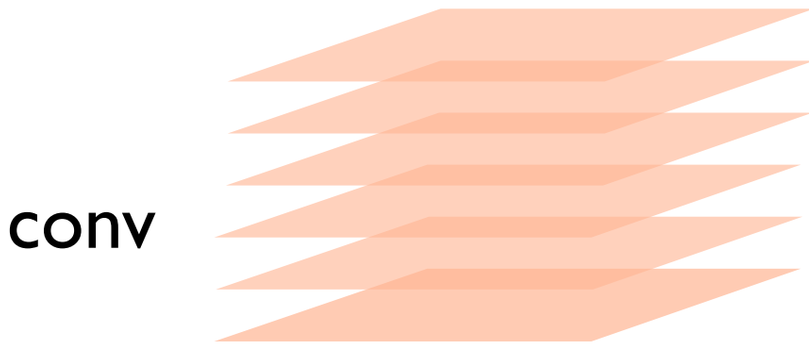
output: probabilities
for 5 directions



input: maze map+
treasure map+
robot pos. map

New concept

output: probability map
for 5 directions across
all of the maze (but use
only at current pos.!)



input: maze map+
treasure map+
robot pos. map

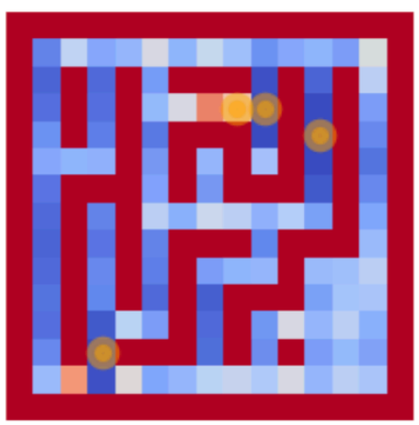
*thanks to Thomas Fösel

seems to work better…

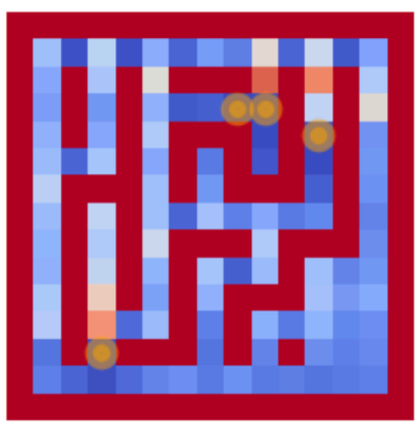
arbitrary mazes
treasure is not deleted

action probability maps for
an arbitrary test map

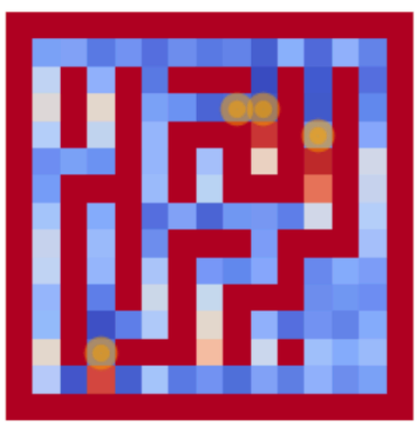
[1 0]



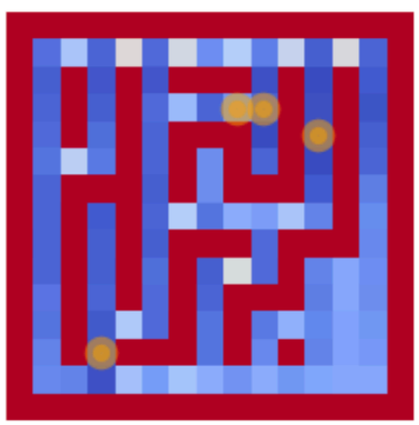
[0 -1]



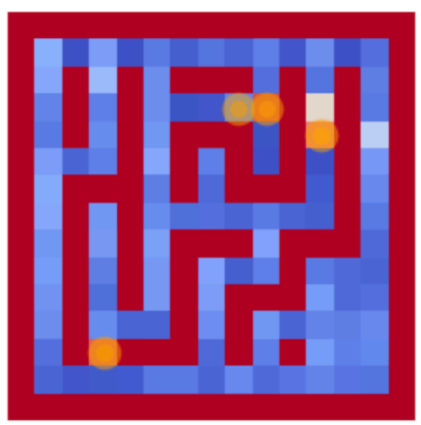
[0 1]



[-1 0]



[0 0]



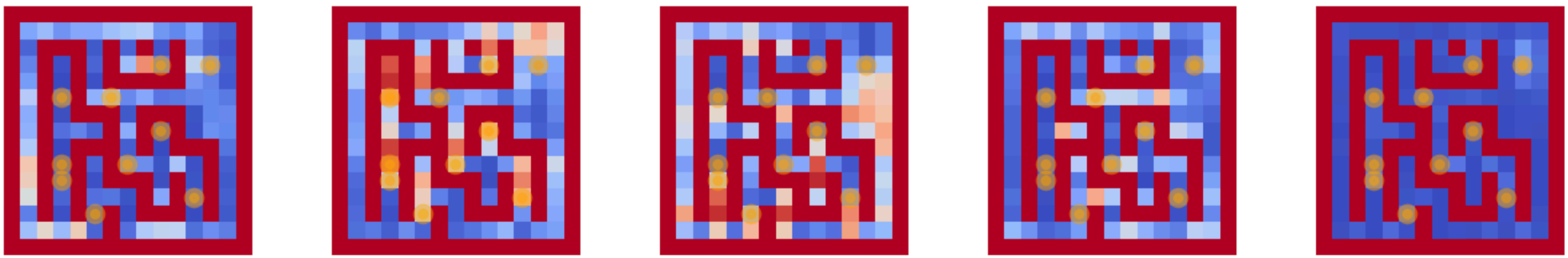
after 600*10 trajectories

arbitrary mazes, treasure
not deleted, 4 chests,
batchsize 10, 30 time-steps,
punish a wall hit: -0.1 reward

arbitrary mazes
treasure is deleted

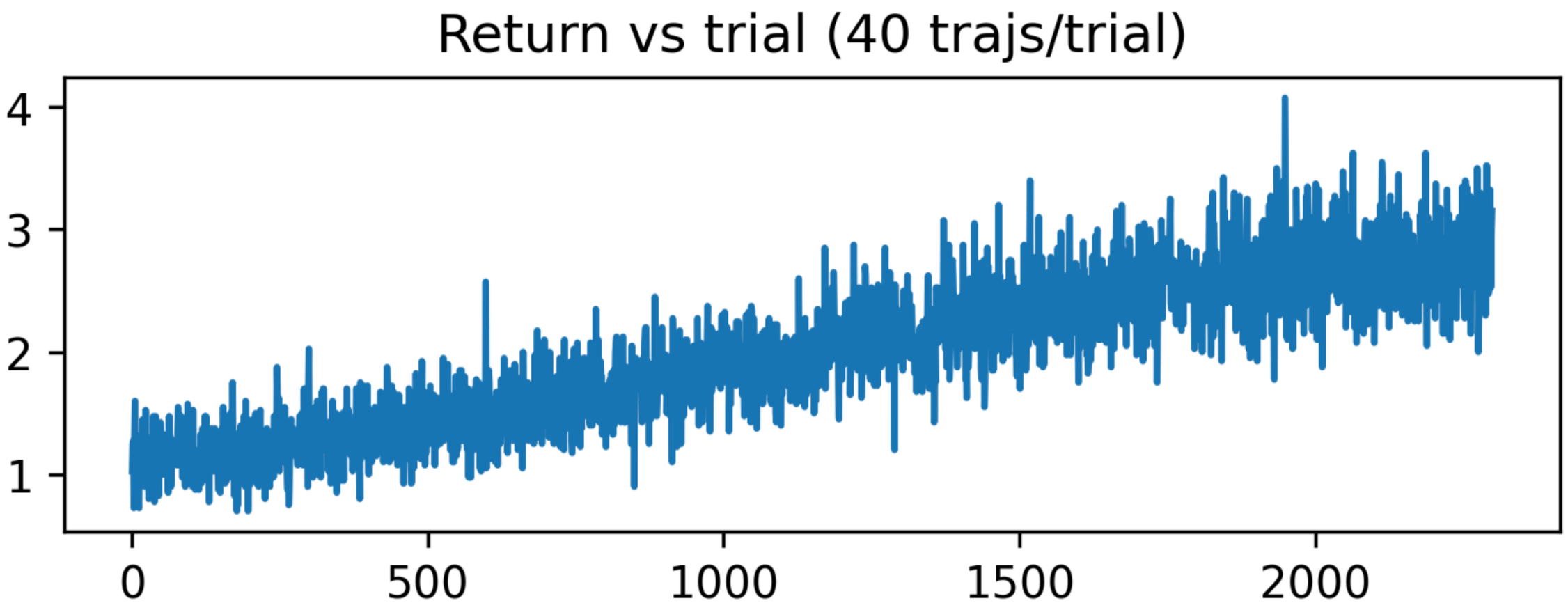
action probability maps for
an arbitrary test map

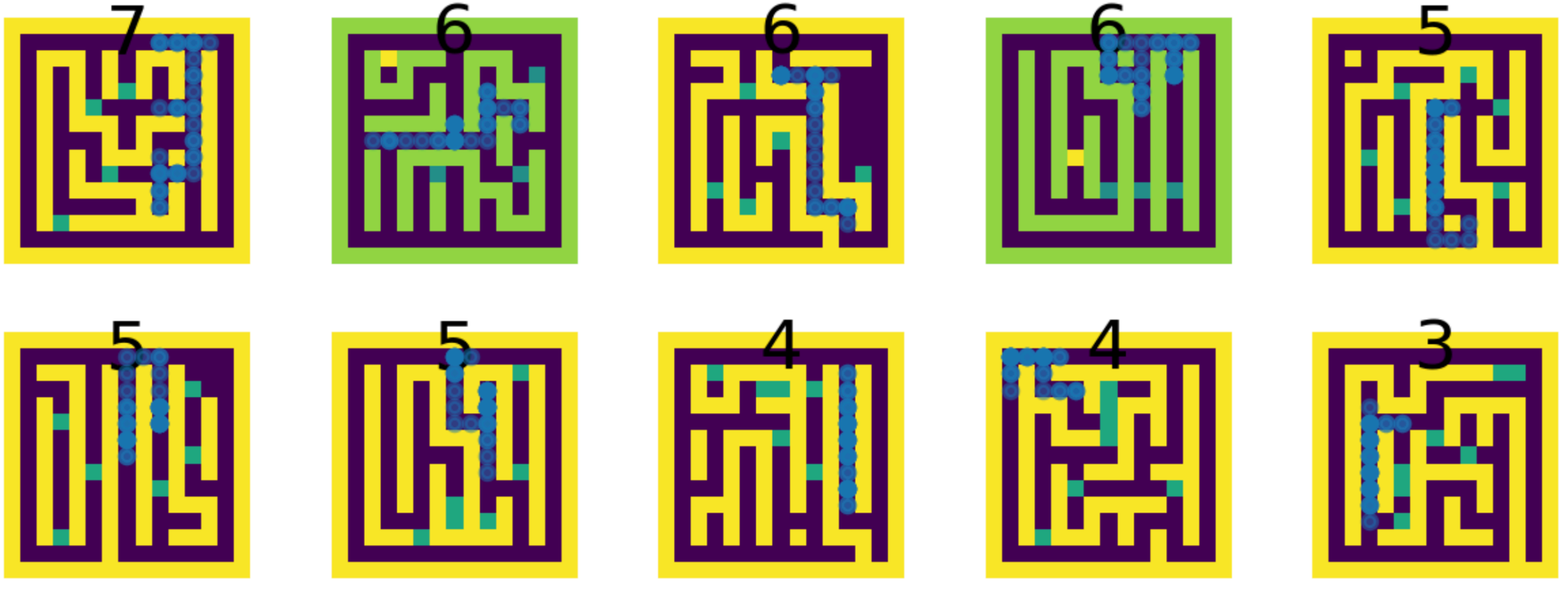
[1 0] [0 -1] [0 1] [-1 0] [0 0]



after 2300*40 trajectories

arbitrary mazes, treasure
is deleted, 10 chests,
batchsize 40, 40 time-steps,
punish a wall hit: none


Return vs trial (40 trajs/trial)
4
3
2
1
0
500
1000
1500
2000


7
6
6
6
5
5
5
4
4
3

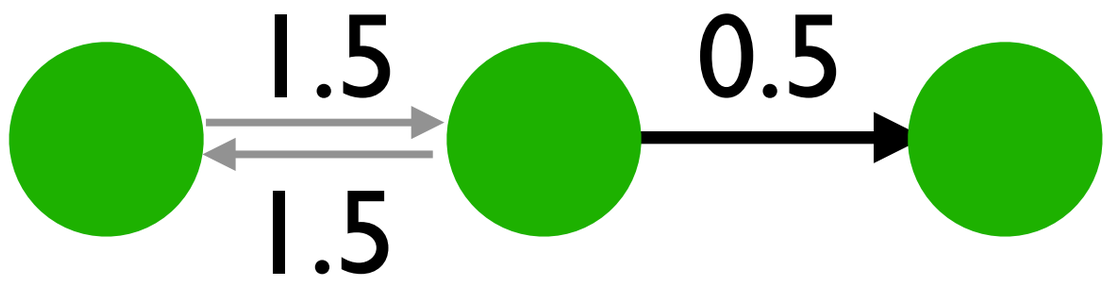
FAU students: please indicate your preferences for the exam date (see the forum)

Machine Learning for Physicists

Quiz

Machine Learning for Physicists

Markov chains



What is the steady state distribution?

a 0.4,0.4,0.2

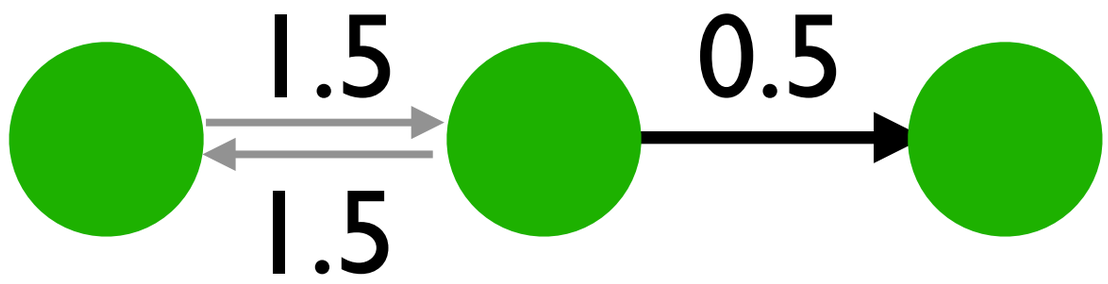
b 0,0,1

c 0.5,0.5,0

d 0.43,0.43,0.14

e 1.5,1.5,0.5

Markov chains



What is the steady state distribution?

a 0.4,0.4,0.2

b 0,0,1

c 0.5,0.5,0

d 0.43,0.43,0.14

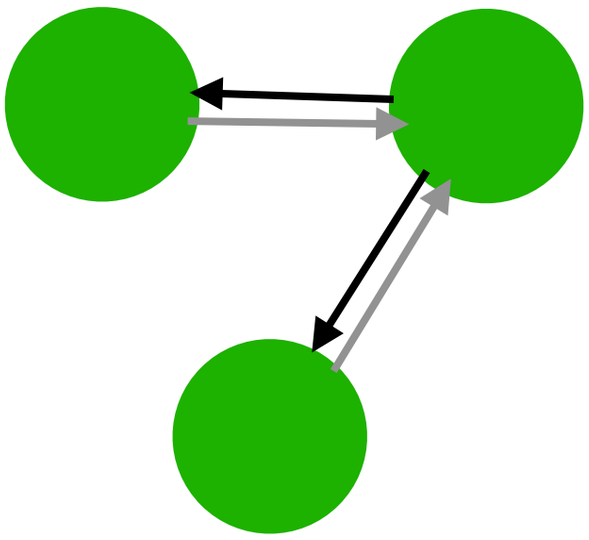
e 1.5,1.5,0.5

Detailed balance

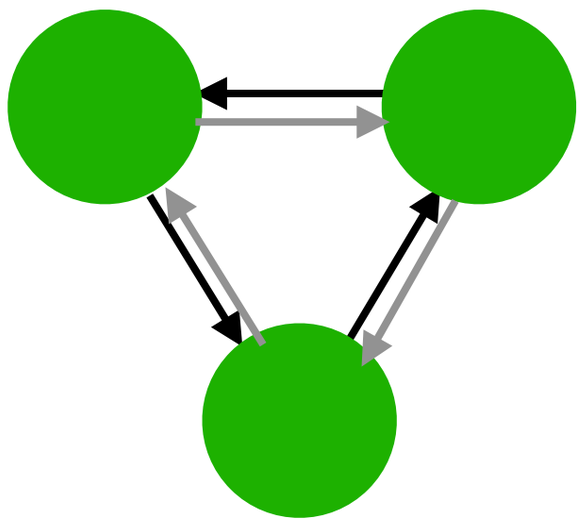
Which one is **not** detailed balance?

[transition rate strength indicated by gray level of arrow]

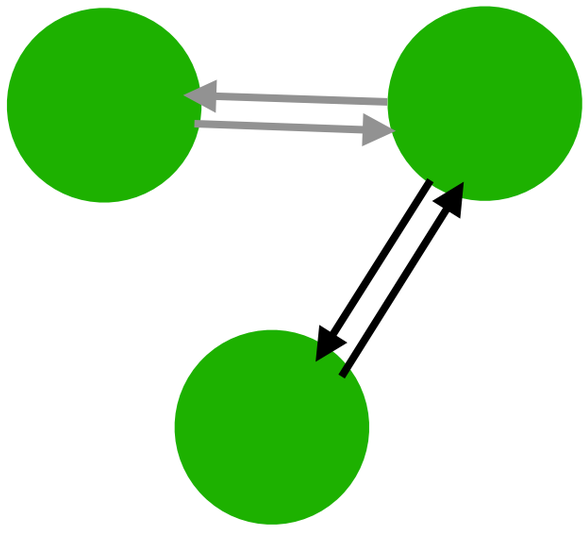
a



b



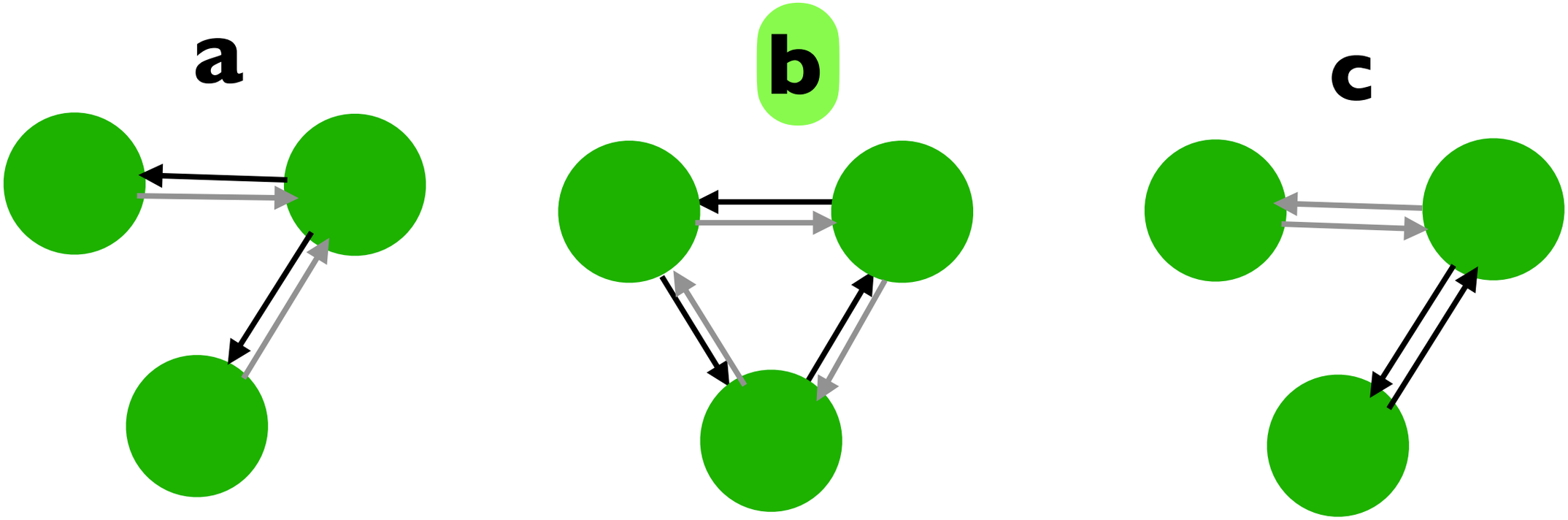
c



Detailed balance

Which one is **not** detailed balance?

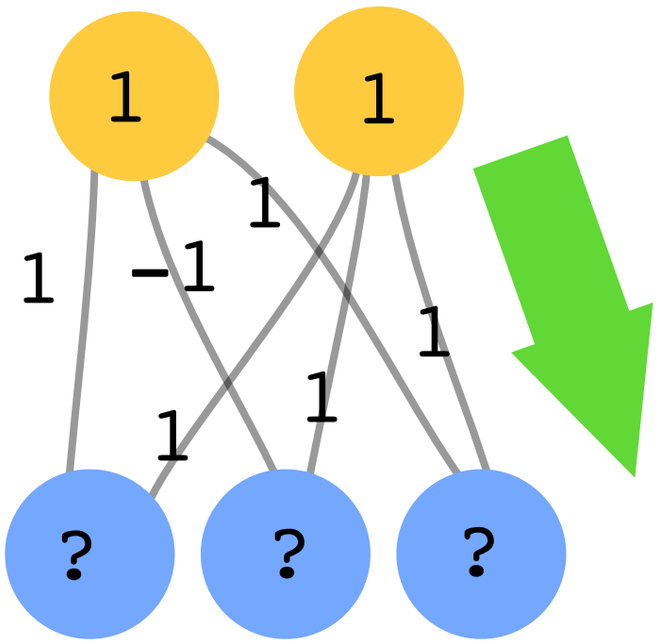
[transition rate strength indicated by gray level of arrow]



Boltzmann Machine

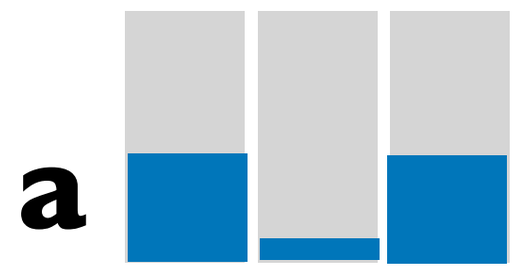
"hidden" units **h**

(all biases = 0)



"visible" units **v**

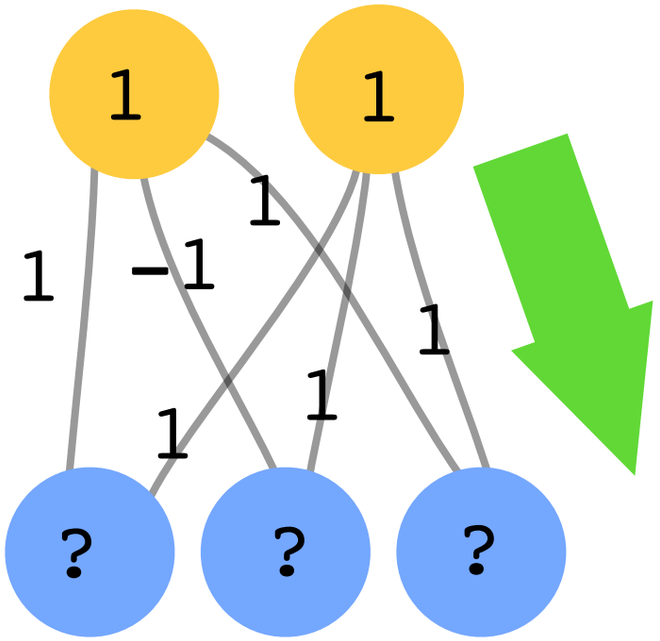
what is the P(v|h) distribution in this step?



Boltzmann Machine

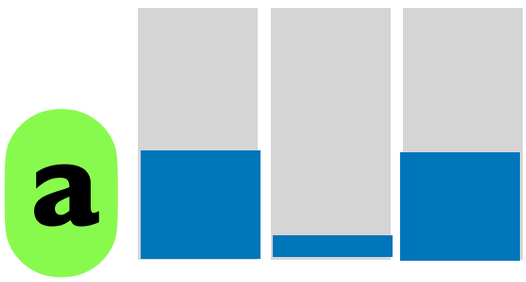
"hidden" units **h**

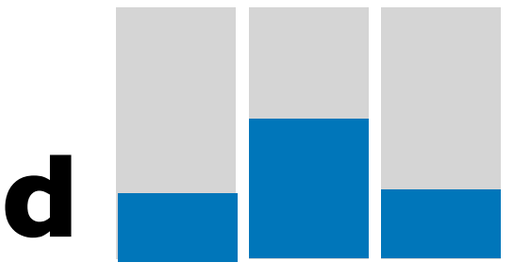
(all biases = 0)



"visible" units **v**

what is the P(v|h) distribution in this step?





Lecture 9 Homework

(will be briefly discussed in the online session for lecture 10)

Machine Learning for Physicists

1 * Apply the Boltzmann machine to the MNIST images!
(explore the role of learning rate, batchsize, number of hidden units, etc.)

2 **Challenge:** Who can get the highest return in the maze reinforcement learning?

[simulation parameters: 15x15 maze, 10 treasure chests, 40 time steps, chests deleted; like in the last example in the 'improved' deep policy gradient notebook]