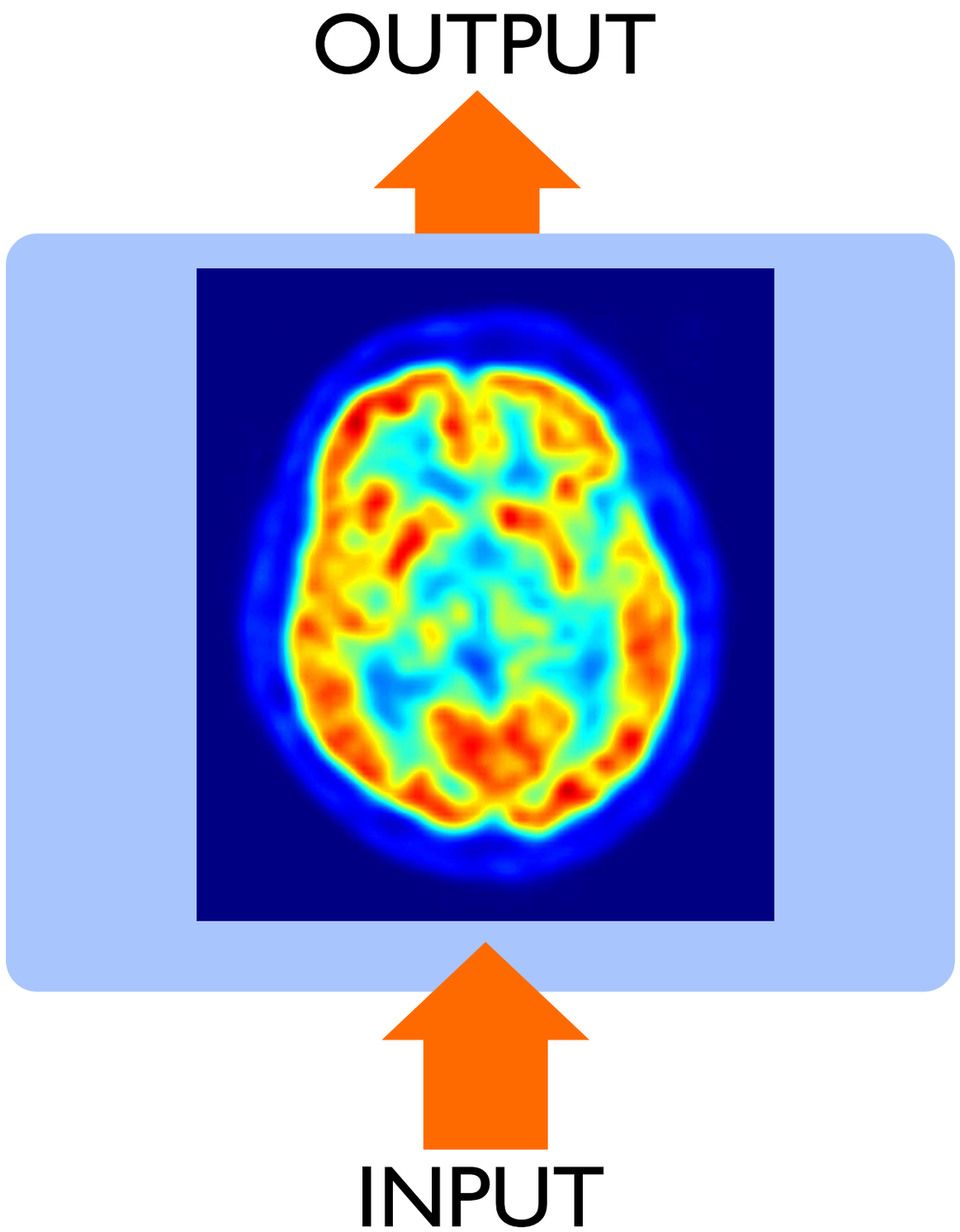
Machine Learning for Physicists
Lecture 1

Summer 2017

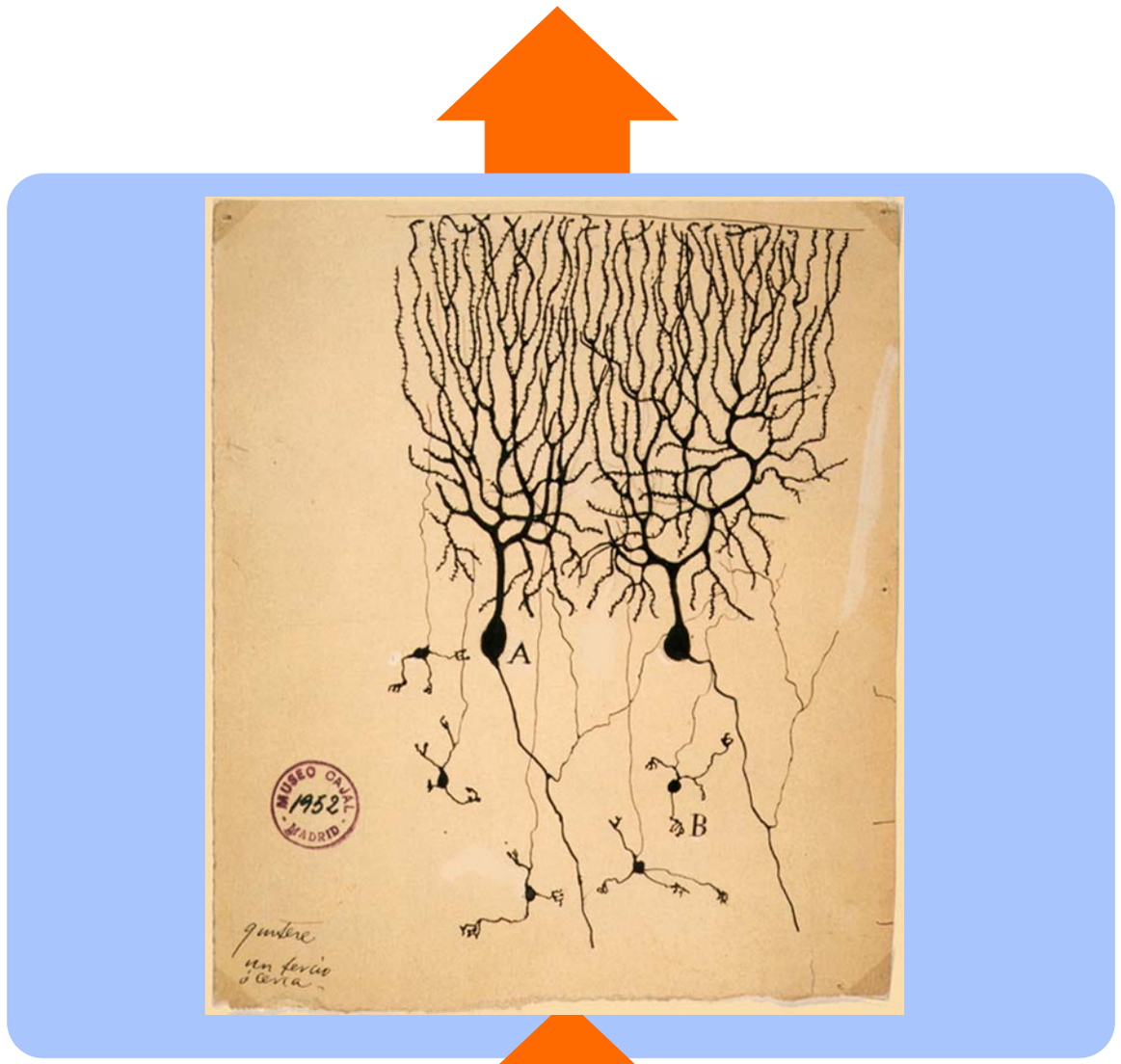
University of Erlangen-Nuremberg
Florian Marquardt

(Image generated by a net with 20 hidden layers)



(Picture: Wikimedia Commons)

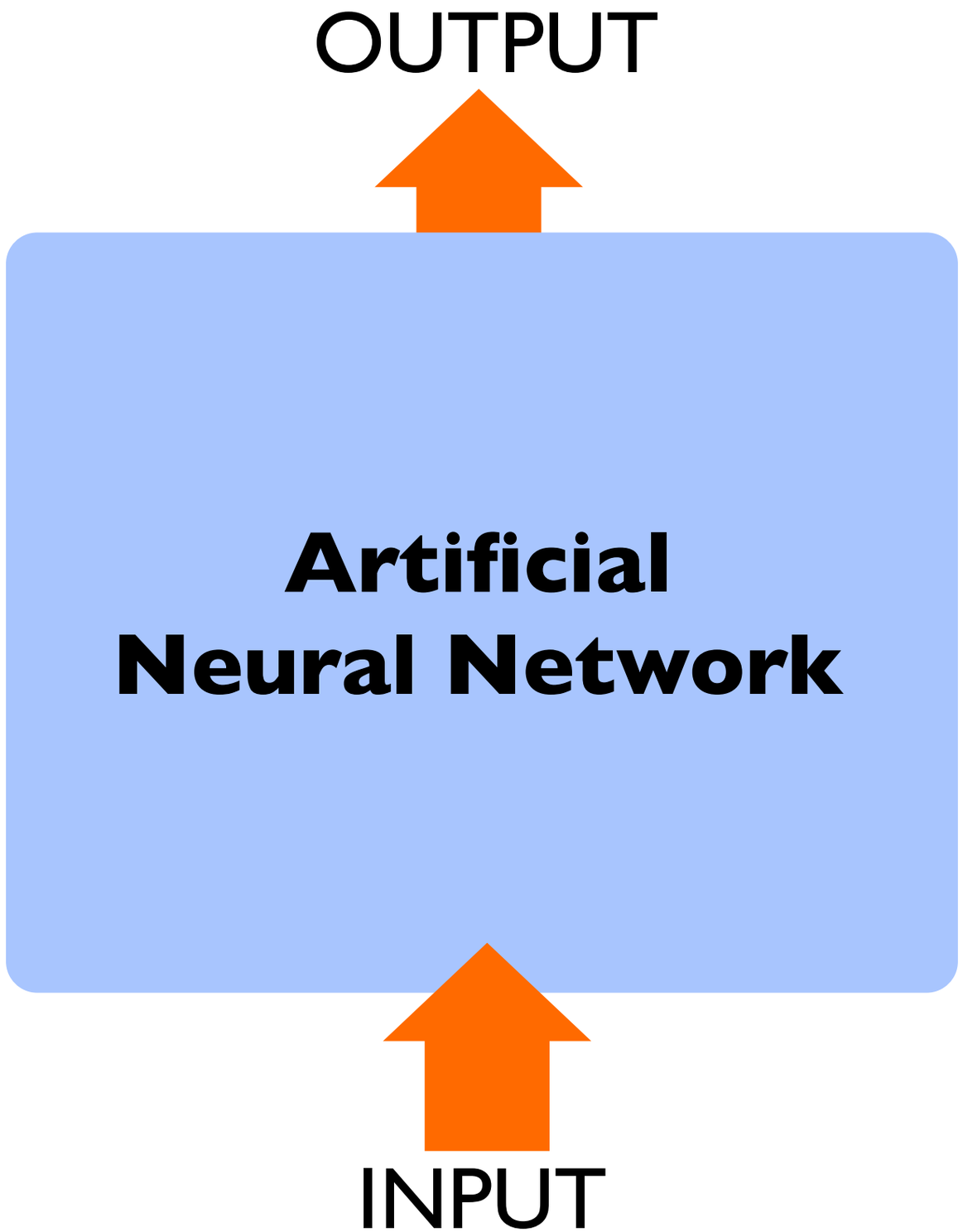
OUTPUT

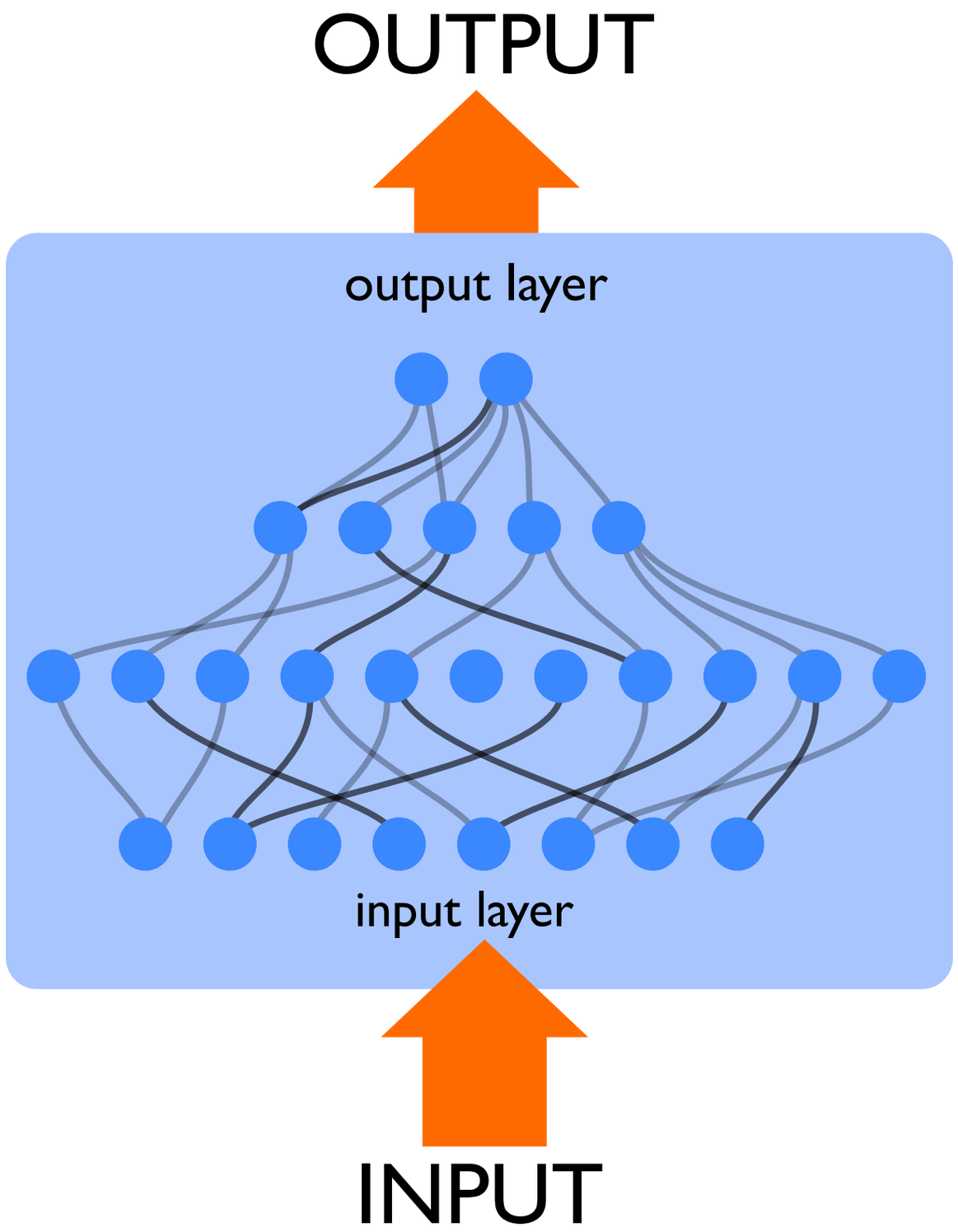


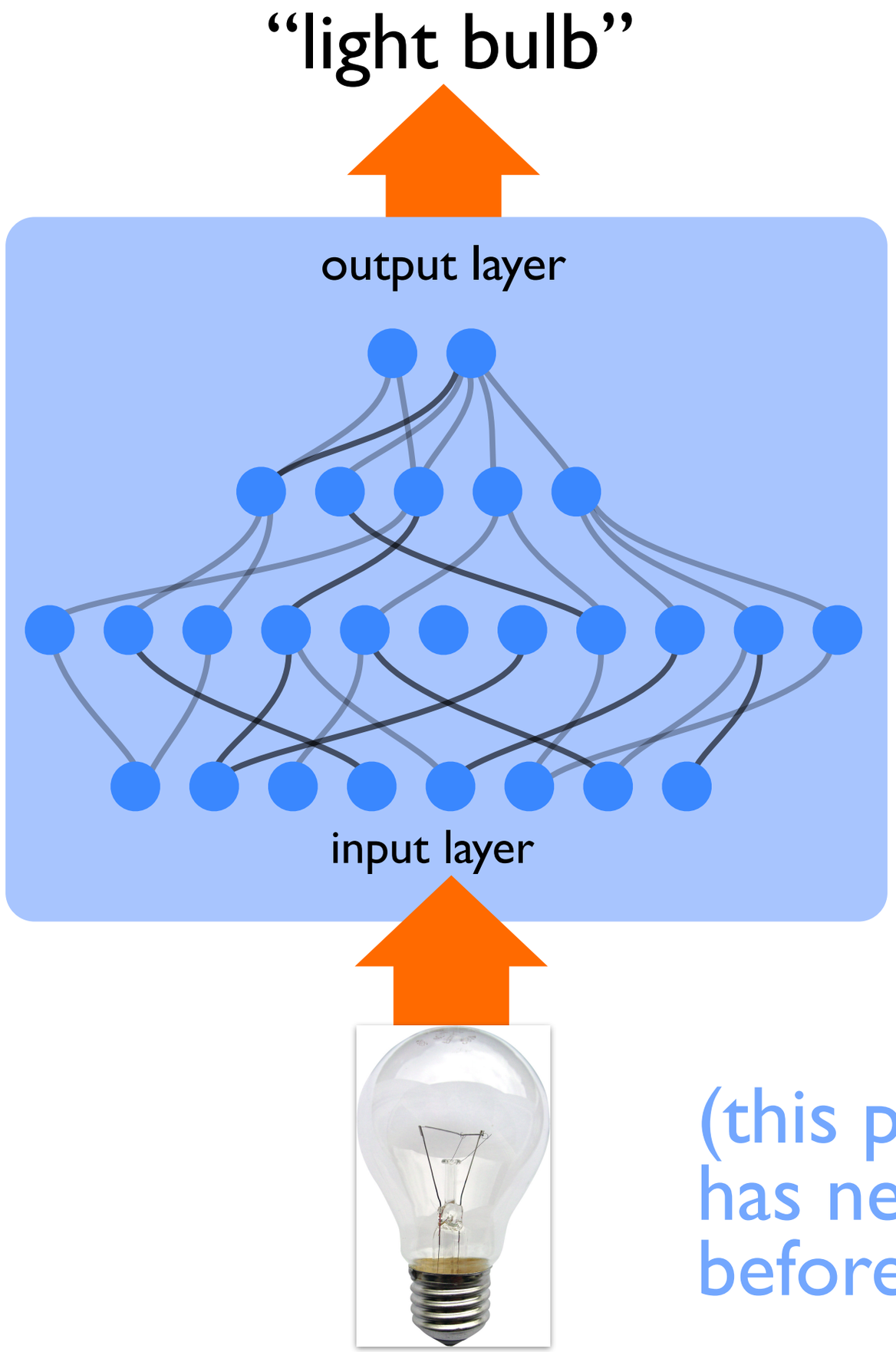
INPUT

(drawing by Ramon y Cajal, ~1900)

(Picture: Wikimedia Commons)

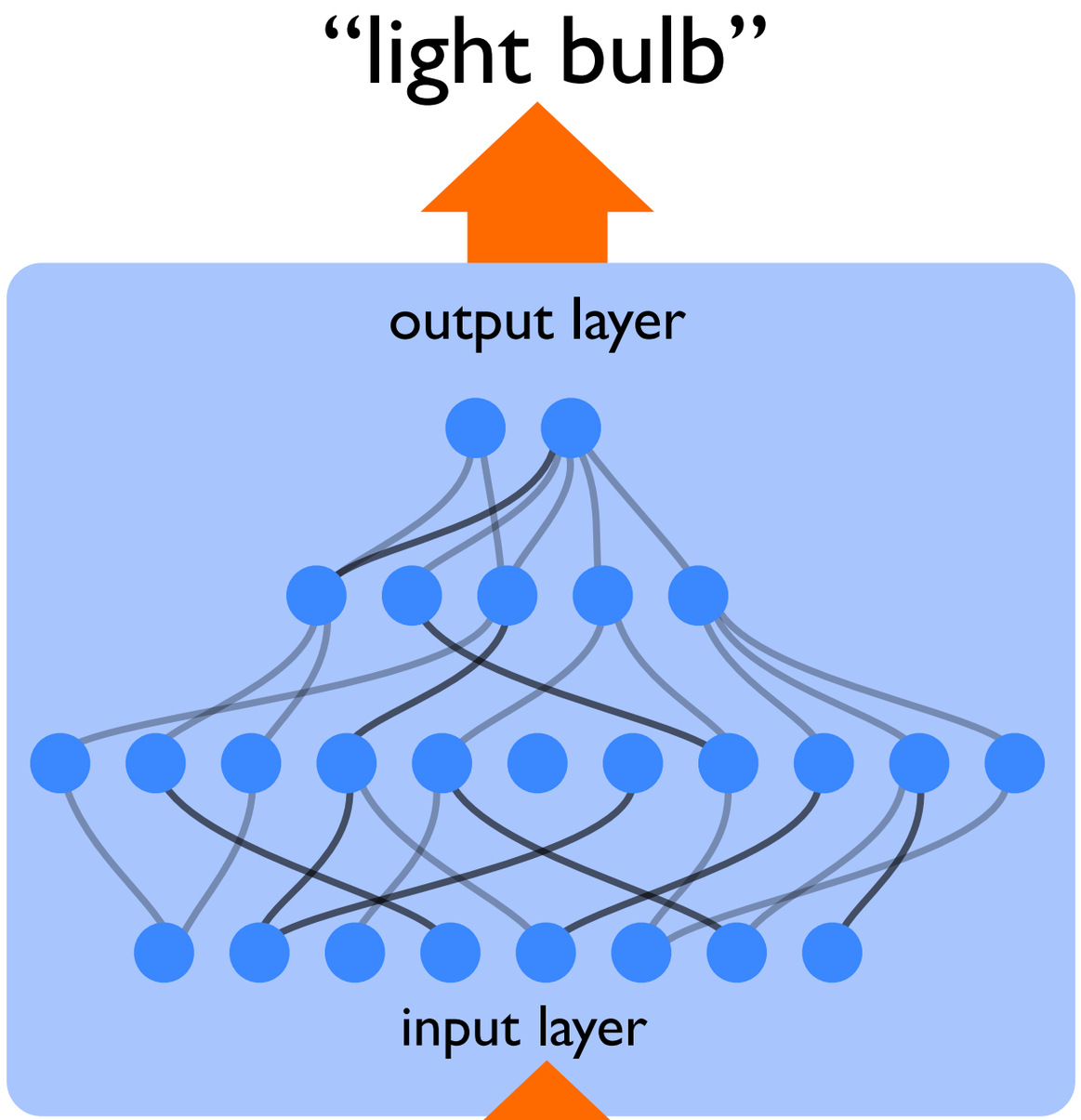

OUTPUT
Artificial
Neural Network
INPUT


OUTPUT
output layer
input layer
INPUT



(this particular picture has never been seen before!)

(Picture: Wikimedia Commons)



(training images)

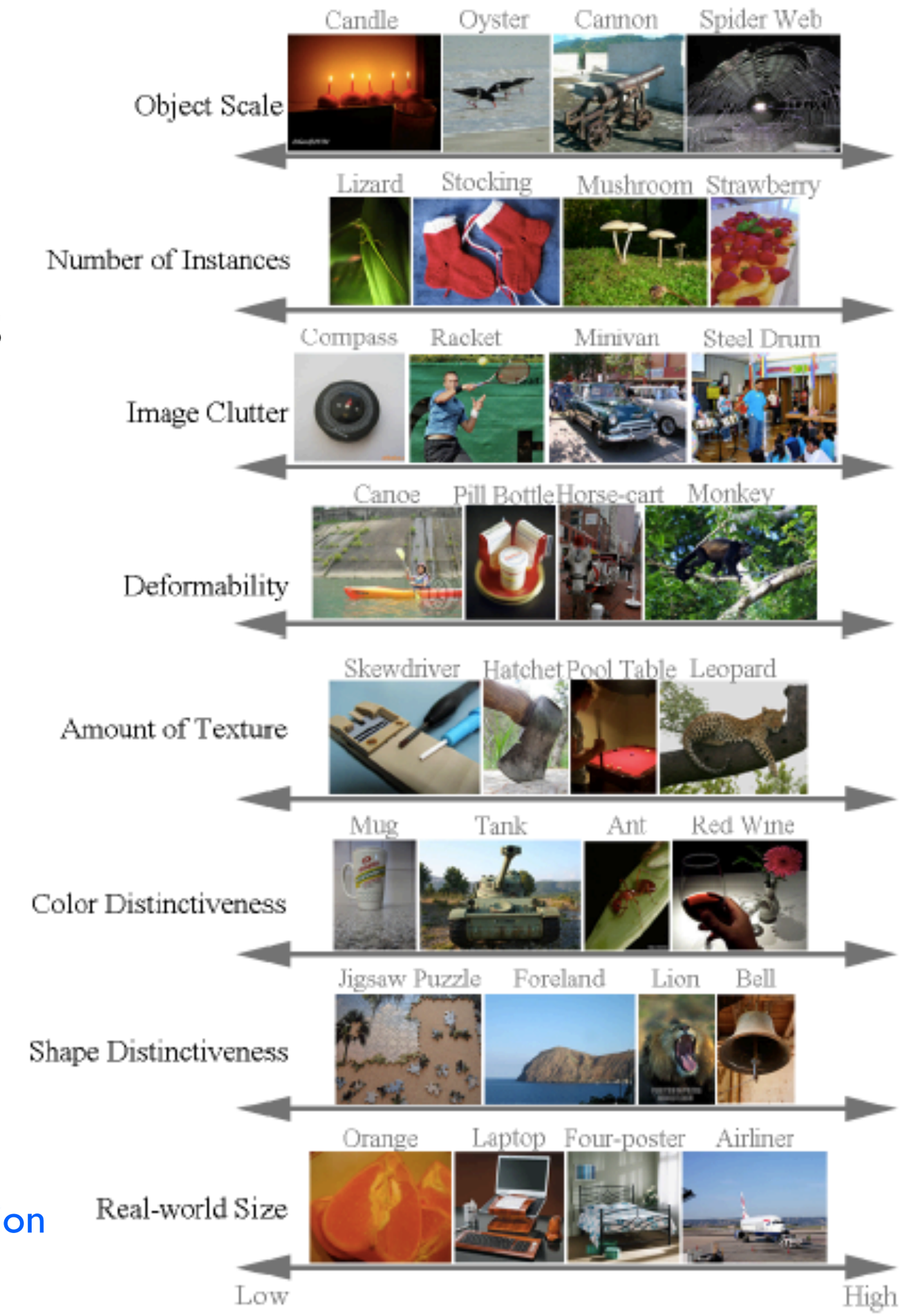



(Picture: Wikimedia Commons)

ImageNet competition

1.2 million training pictures (annotated by humans)
1000 object classes

2012: A deep neural network beats competition clearly (16% error rate; since then rapid decrease of error rate, down to about 7%)



Picture: "ImageNet Large Scale Visual Recognition Challenge", Russakovsky et al. 2014

Example applications of (deep) neural networks

(see links on website)

e.g. http://machinelearningmastery.com/inspirational-applications-deep-learning/

Recognize images
Describe images in sentences
Colorize images
Translate languages (French to Spanish, etc.)
Answer questions about a brief text
Play video games & board games at superhuman level

(in physics:)
predict properties of materials
classify phases of matter
represent quantum wave functions

Lectures Outline

- Basic structure of artificial neural networks
- Training a network (backpropagation)
- Exploiting translational invariance in image processing (convolutional networks)
- Unsupervised learning of essential features (autoencoders)
- Learning temporal data, e.g. sentences (recurrent networks)
- Learning a probability distribution (Boltzmann machine)
- Learning from rare rewards (reinforcement learning)
- Further tricks and concepts
- Modern applications to physics and science

Lecture

Learning by doing!

- Basic structure [illegible] neural networks
- Training a [illegible] (backpropagation)

Python

- Expl[illegible]ational invariance in image processing ([illegible]nal networks)
- [illegible]ervised learning of essential featu[illegible] (autoencoders)
- Learning temporal data, e.g. sentences ([illegible] networks)

Keras package for Python

- Learning a probability distribution (Boltzmann machine)
- Learning from rare rewards (reinforcement learning)
- Further tricks and concepts
- Modern applications to physics and science

Homework

Homework: (usually) explore via programming

We provide feedback if desired

No regular tutorial sessions

http://www.thp2.nat.uni-erlangen.de/index.php/2017_Machine_Learning_for_Physicists,_by_Florian_Marquardt

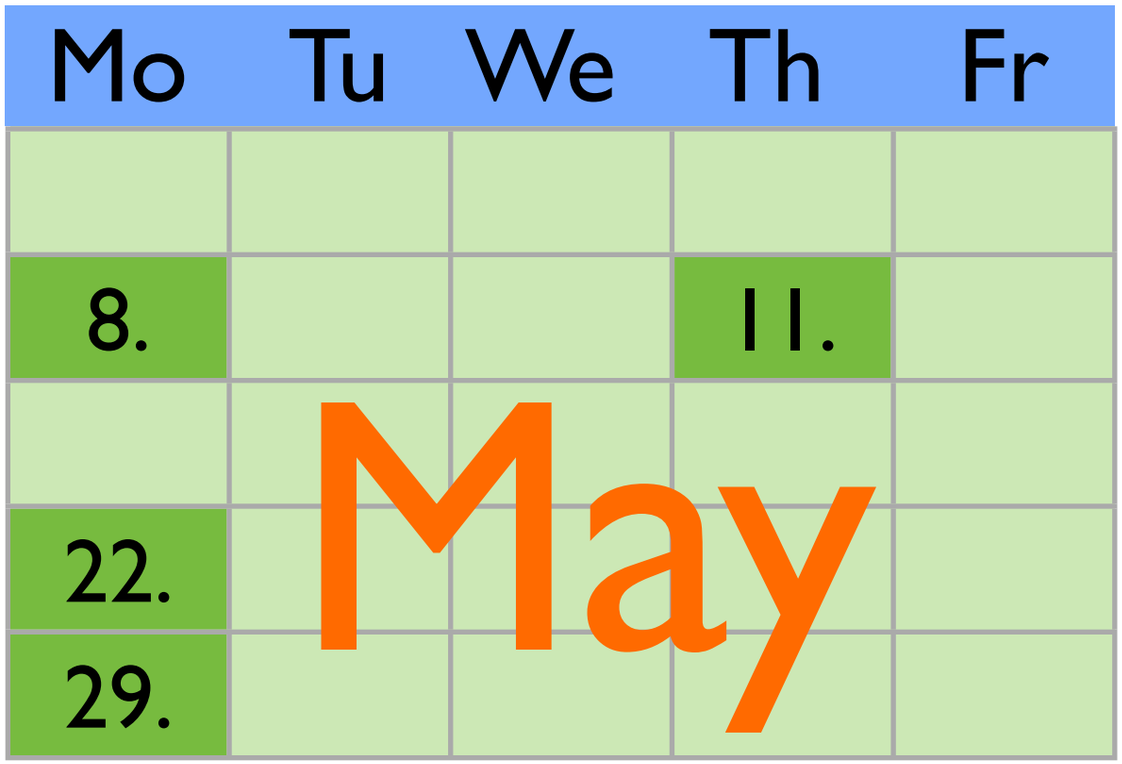
Homework

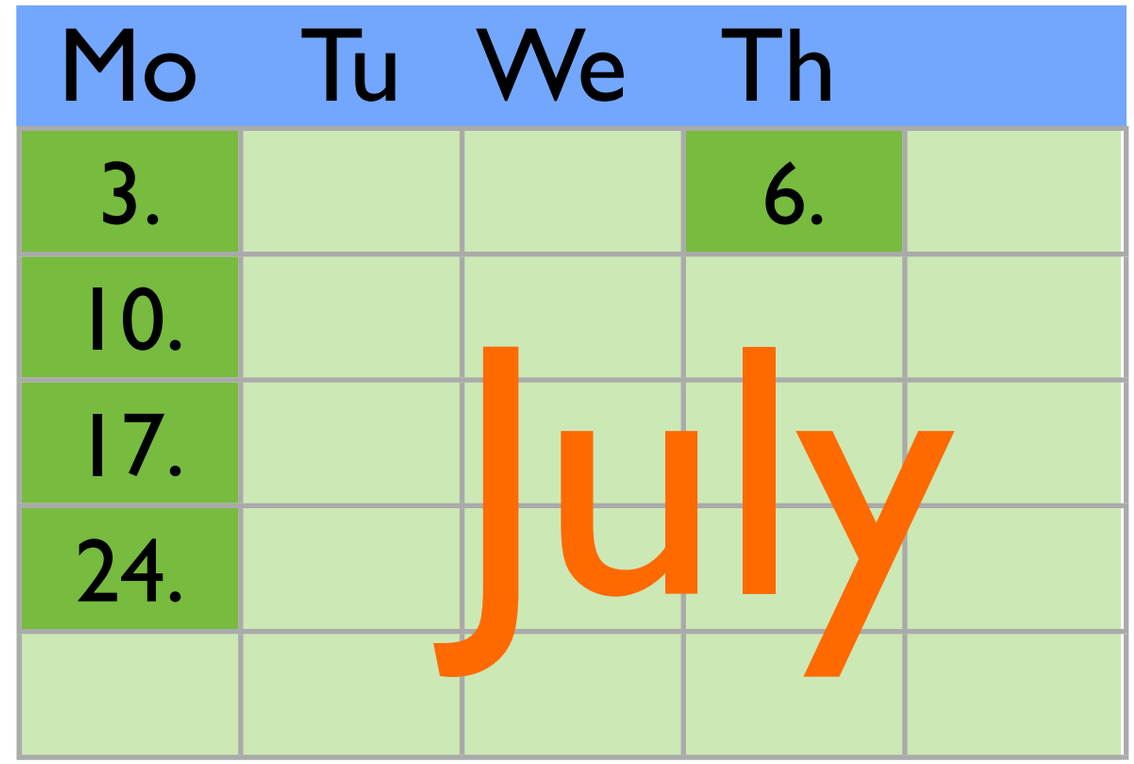
First homework:

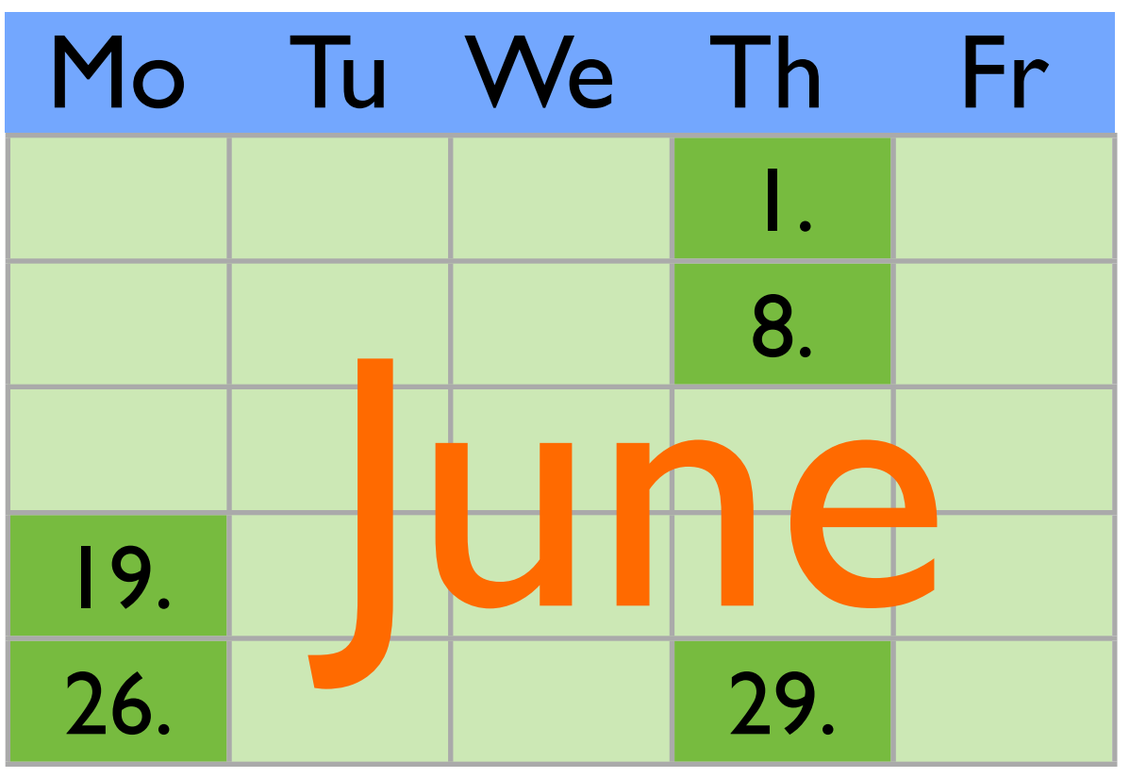
1. Install python & keras on your computer (see lecture homepage); questions will be resolved after second lecture THIS IS IMPORTANT!
2. Brainstorm: "Which problems could you address using neural networks?"

Next time: "installation party" after the lecture

Bring your laptop, if available (or ask questions)



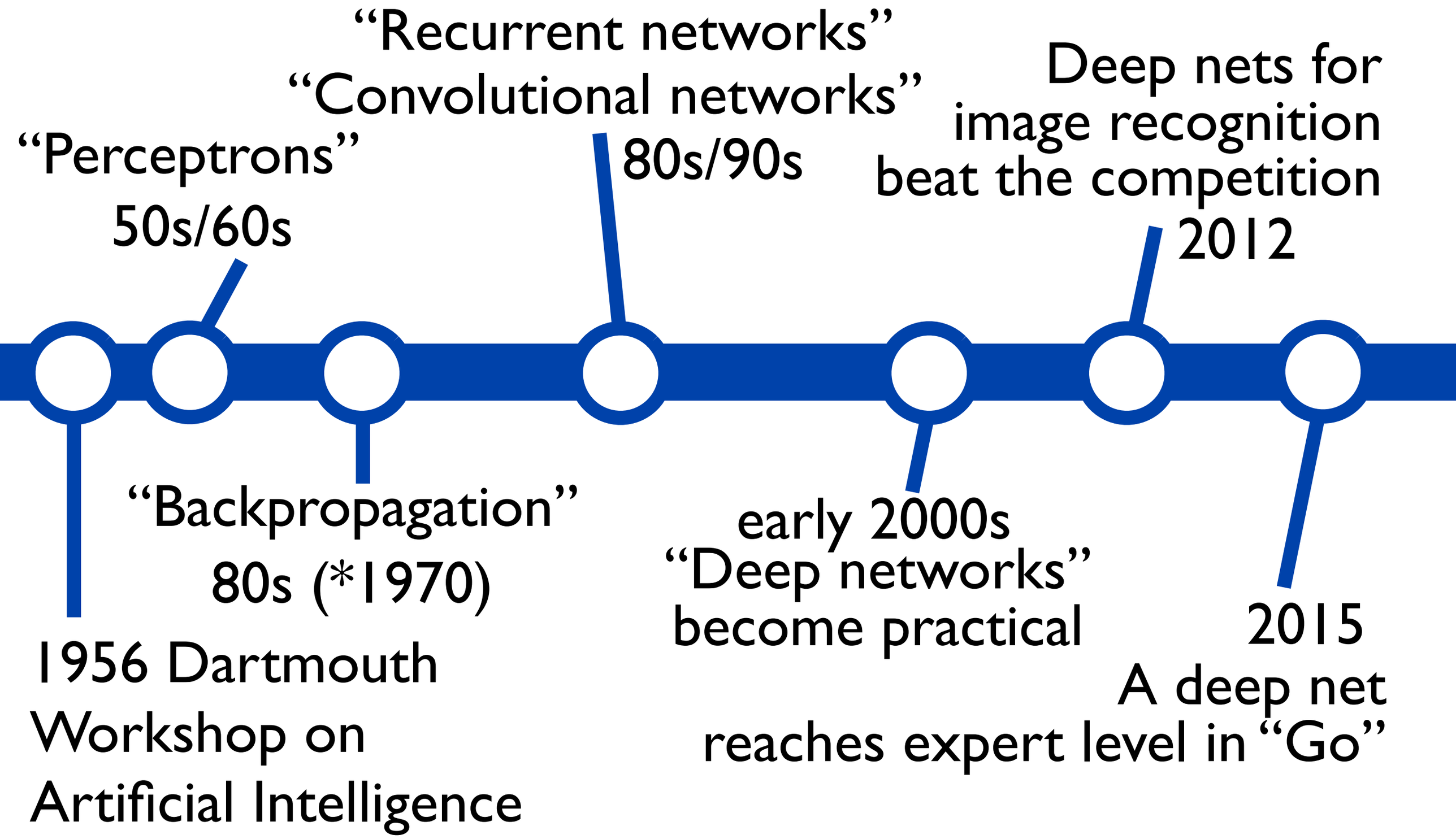




Machine Learning Lecture Schedule (Summer 2017)

Mo/Th 18-20

Very brief history of artificial neural networks



Lots of tutorials/info on the web...

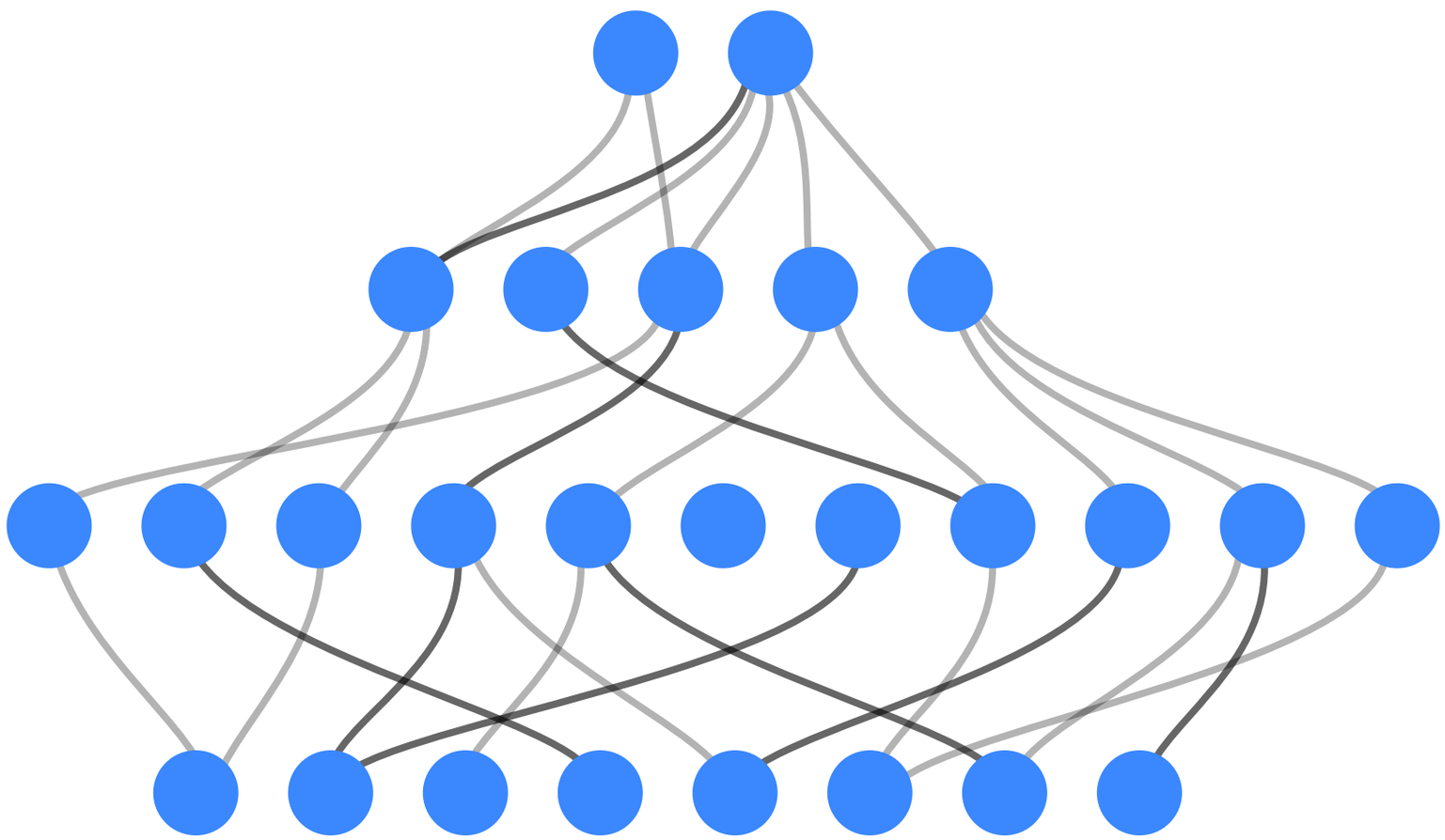
recommend:
online book by Nielsen ("**Neural Networks and Deep Learning**") at https://neuralnetworksanddeeplearning.com

much more detailed book:
"**Deep Learning**" by Goodfellow, Bengio, Courville; MIT press; see also http://www.deeplearningbook.org

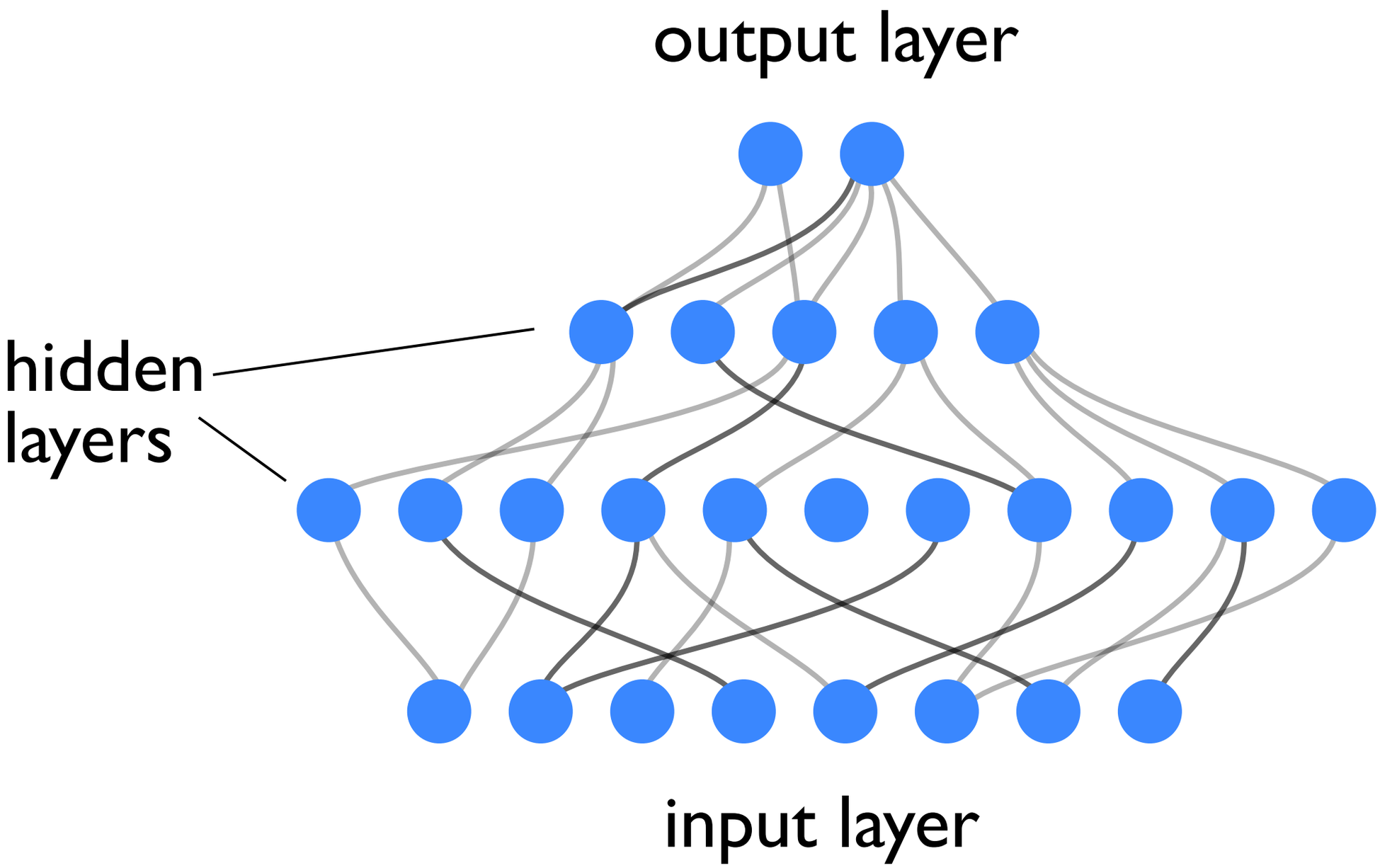
Software –
here: python & keras (builds on theano)

A neural network

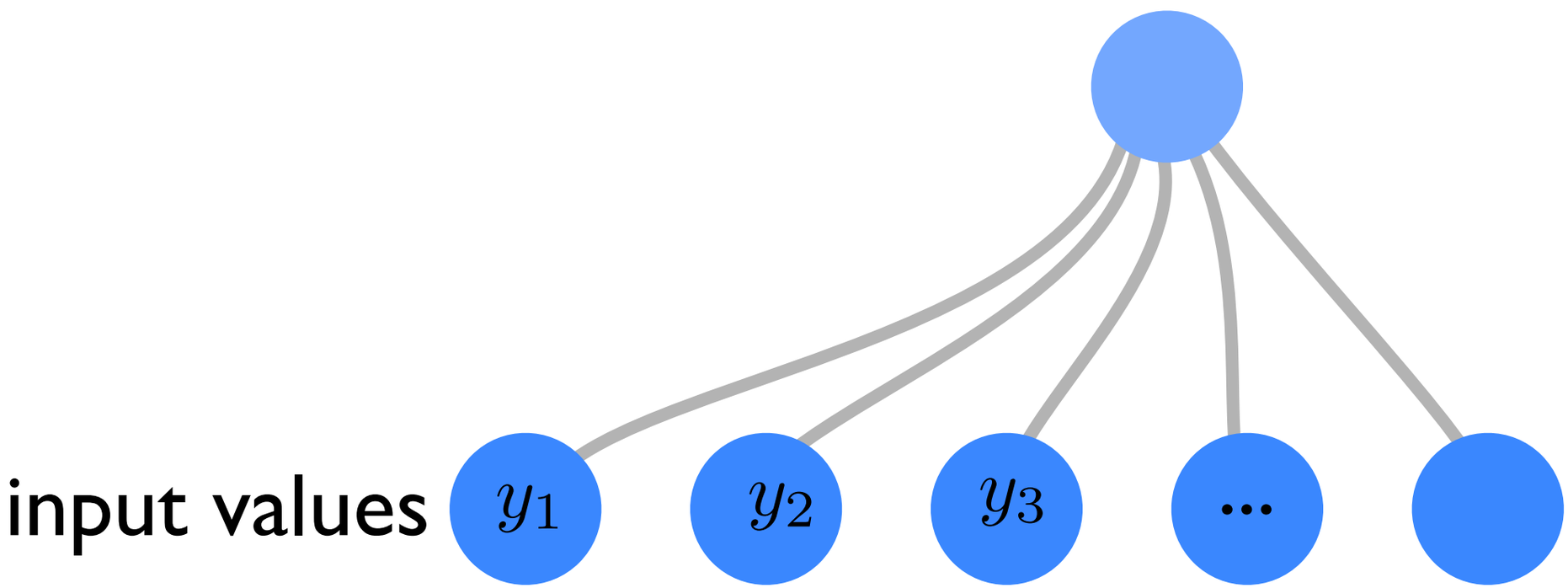
= a nonlinear function (of many variables) that depends on many parameters



A neural network



output of a neuron =
nonlinear function of
weighted sum of inputs

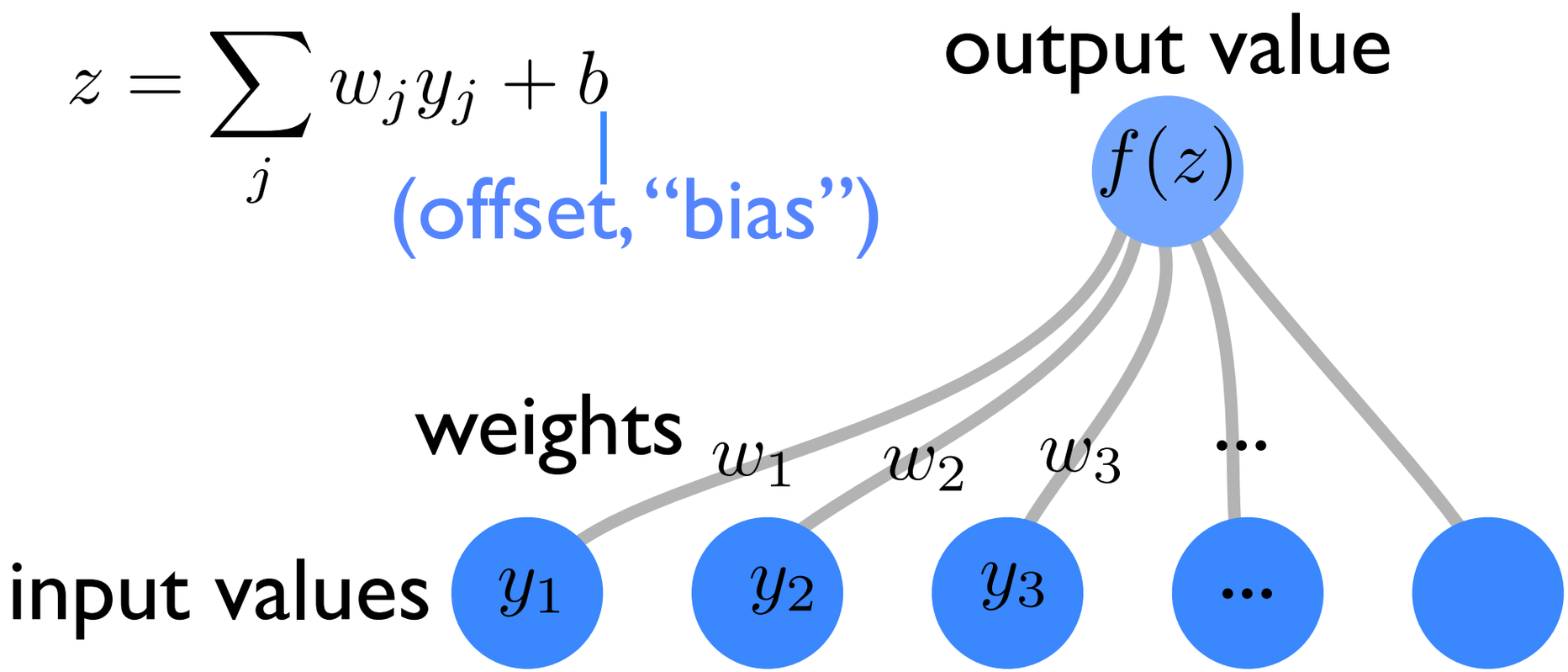


output of a neuron =
nonlinear function of
weighted sum of inputs

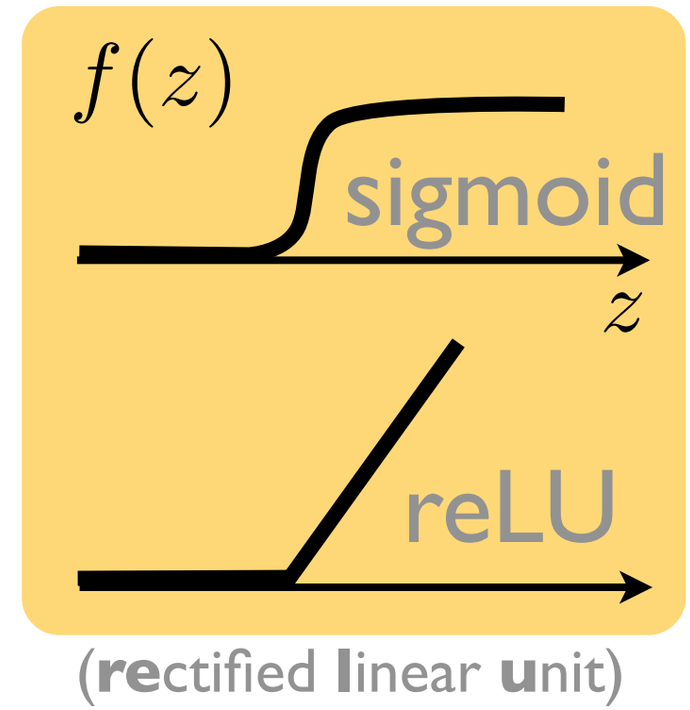
weighted sum

$$z = \sum_j w_j y_j + b$$

(offset, "bias")



output of a neuron =
nonlinear function of
weighted sum of inputs



weighted sum

$$z = \sum_j w_j y_j + b$$

(offset, "bias")

output value

$f(z)$

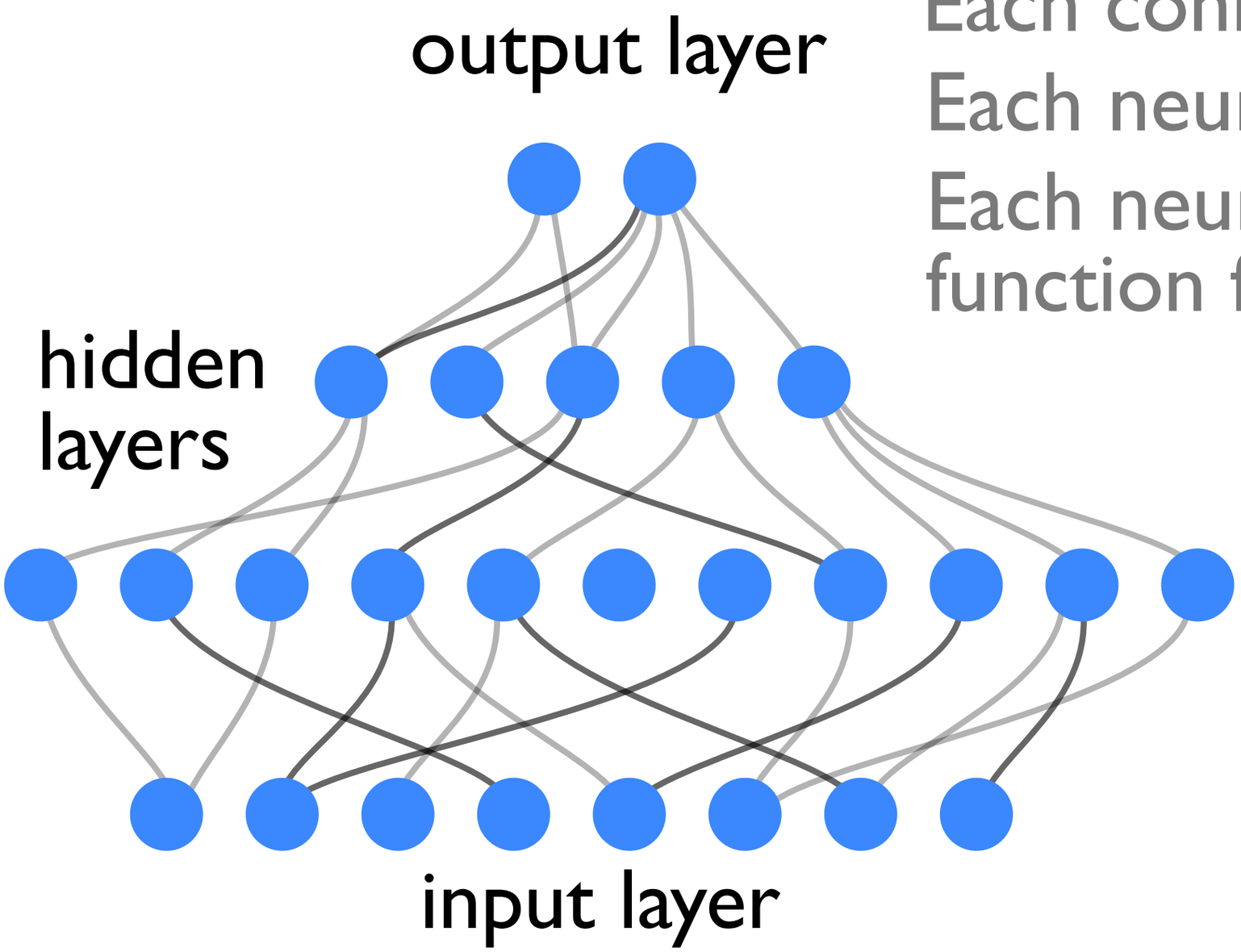
weights

w_1 w_2 w_3 ...

input values

y_1 y_2 y_3 ...

A neural network

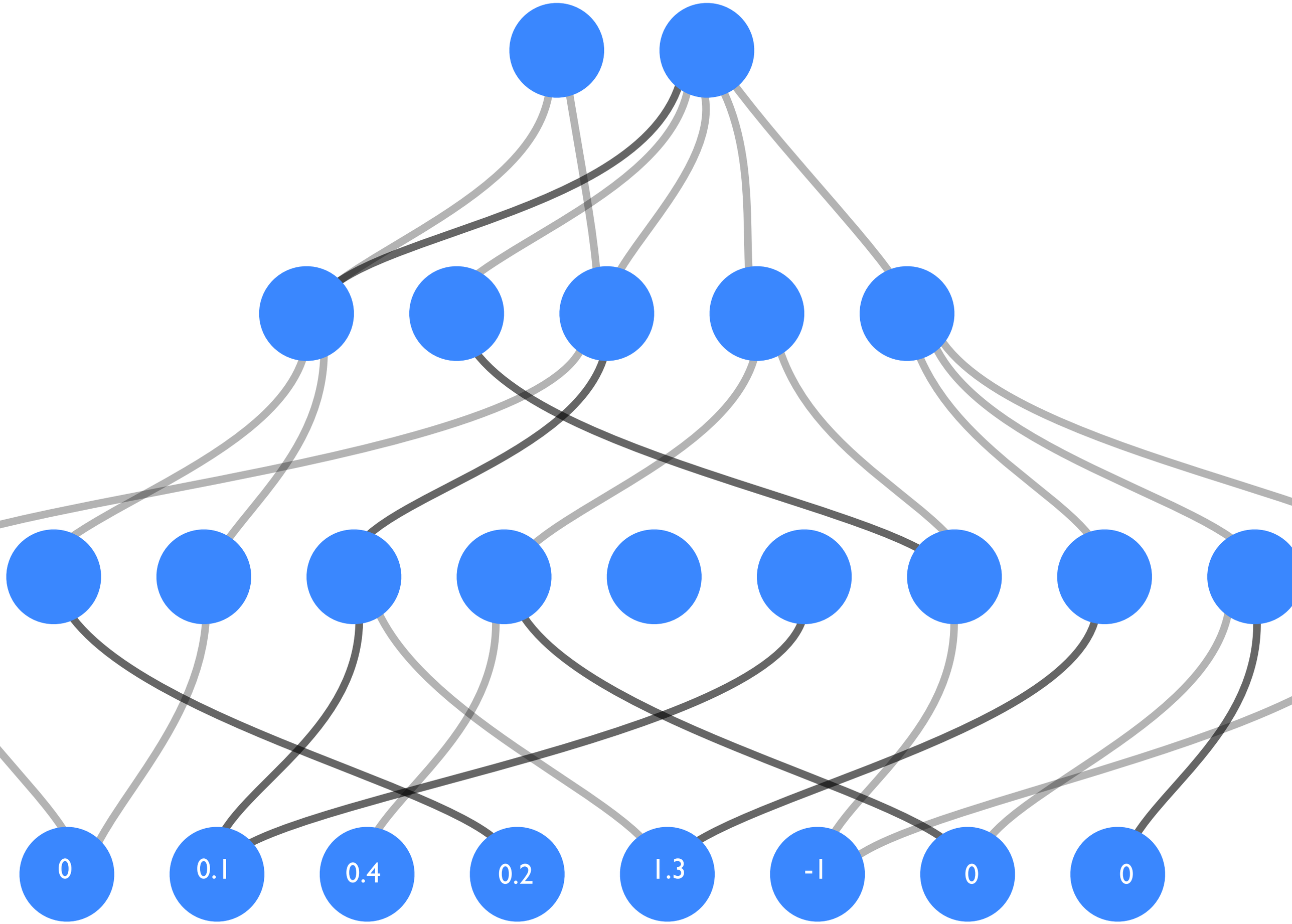


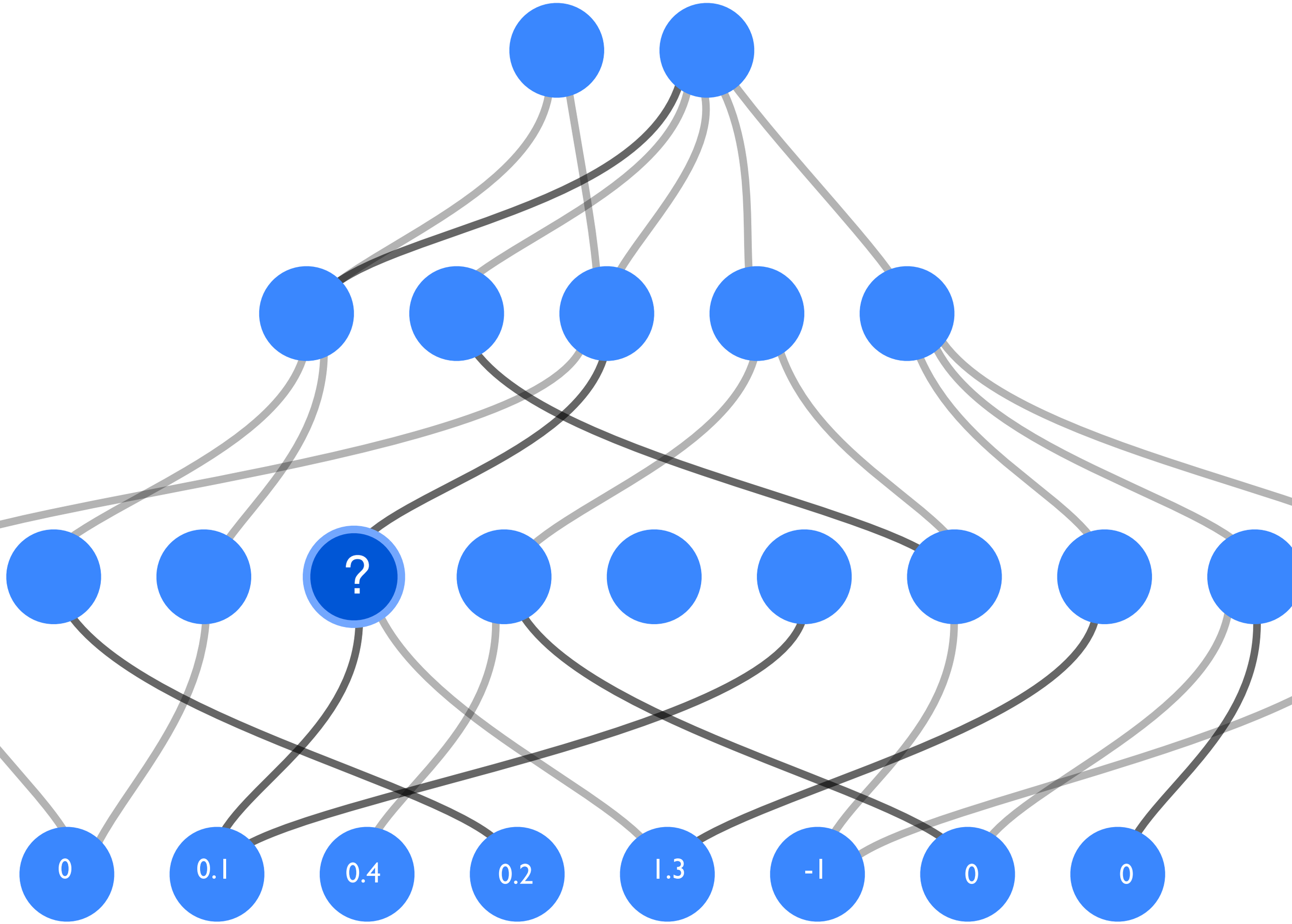
Each connection has a weight w

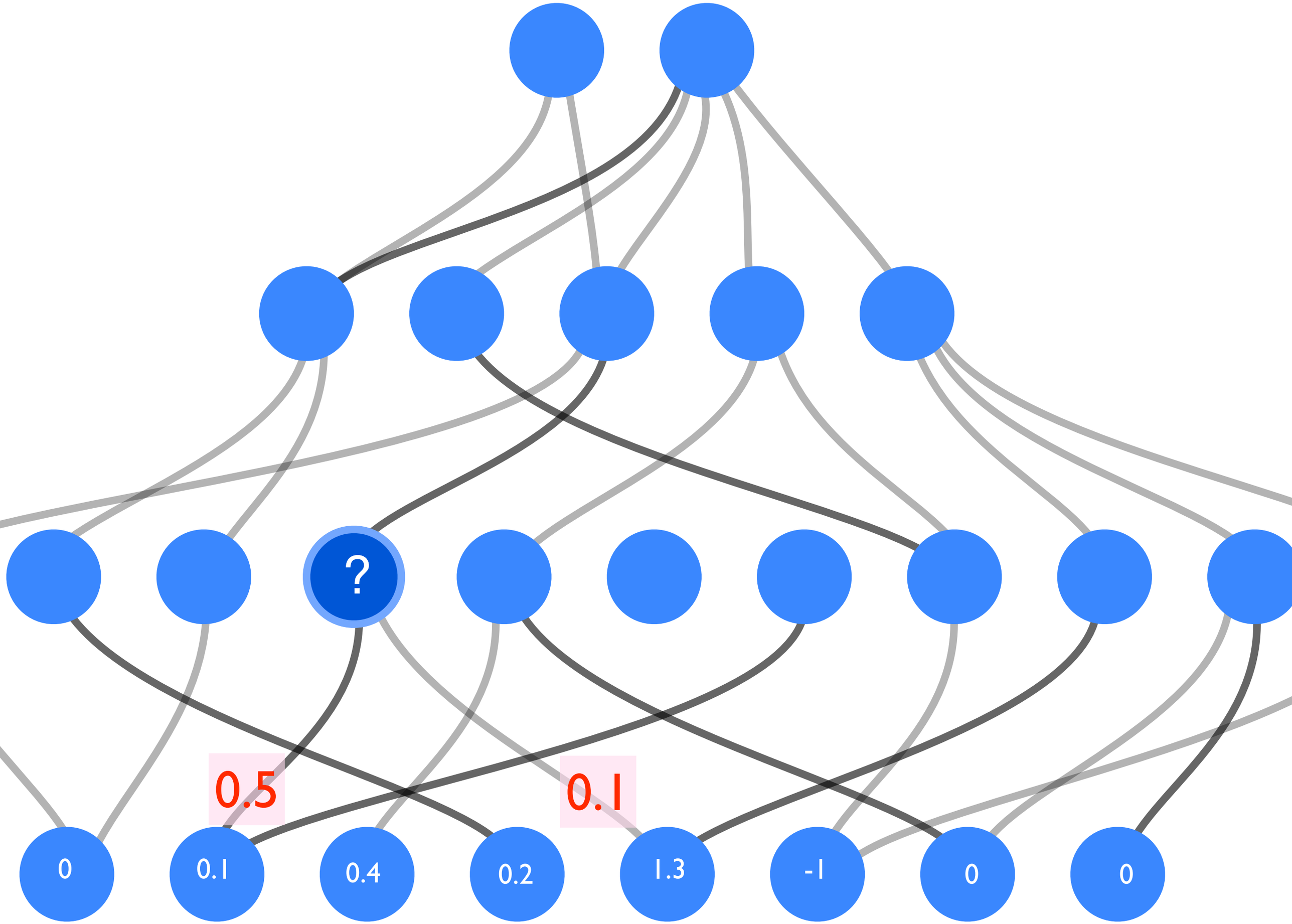
Each neuron has an offset b

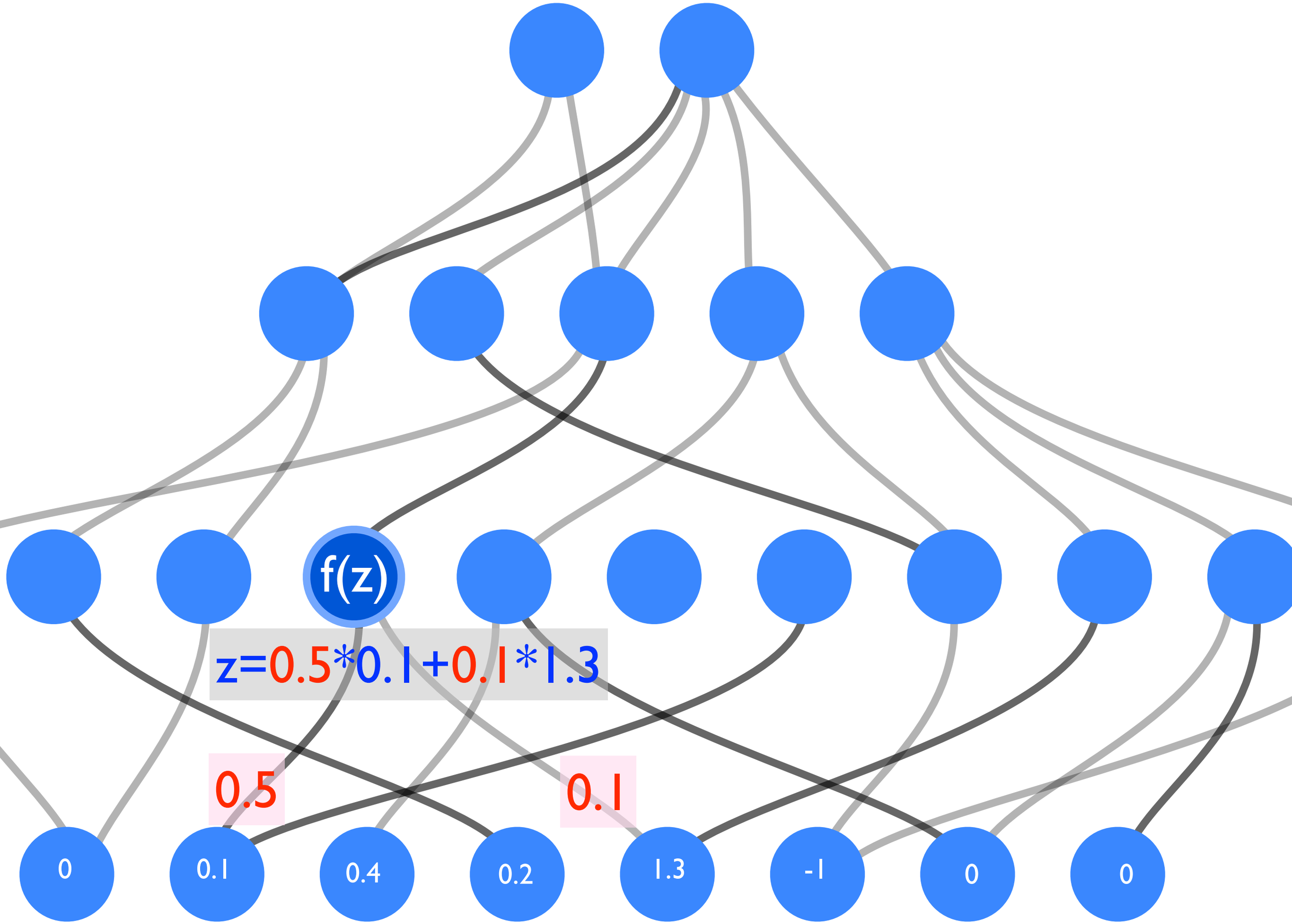
Each neuron has a nonlinear function f (fixed)

The values of input layer neurons are fed into the network from the outside

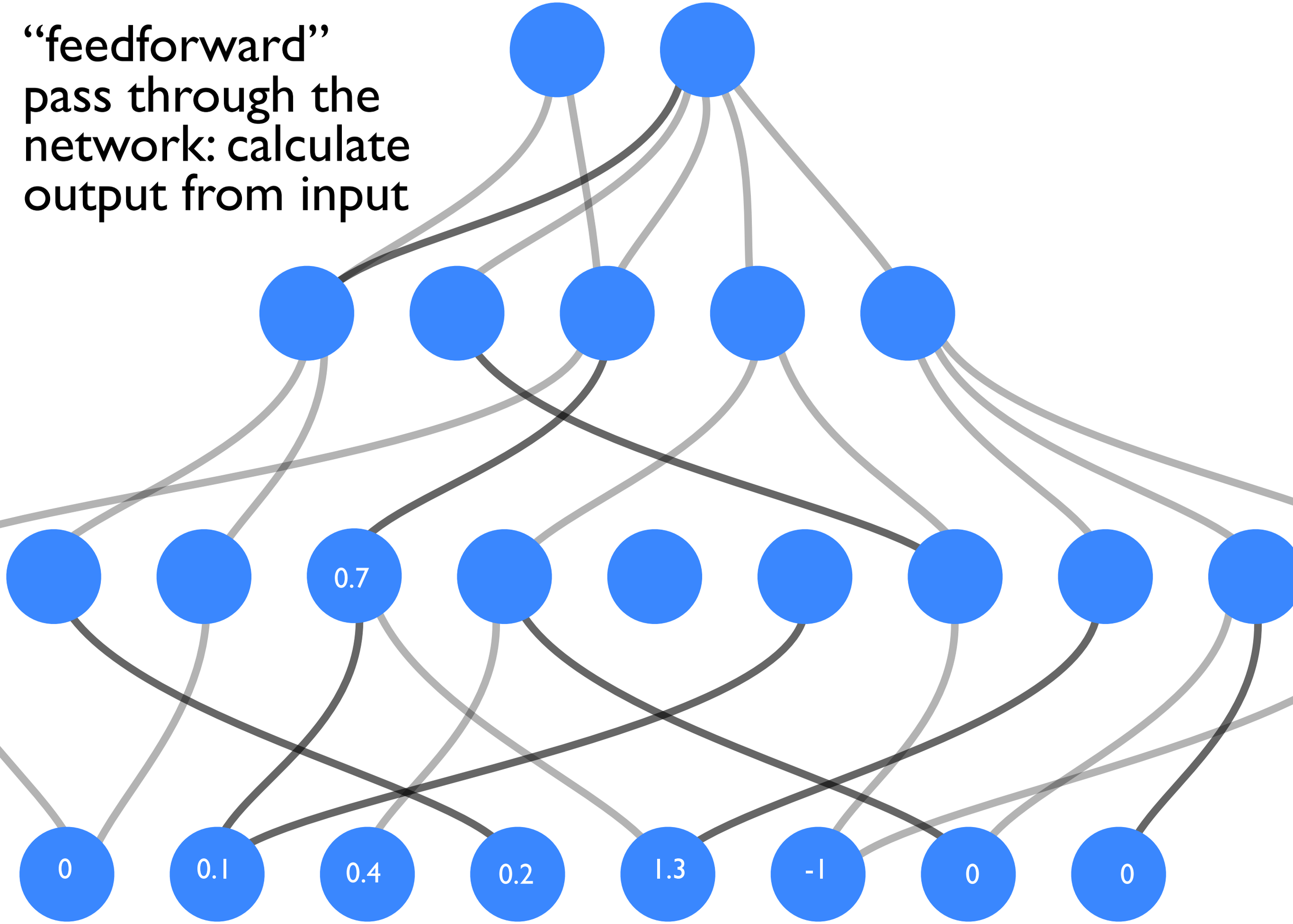

0
0.1
0.4
0.2
1.3
-1
0
0


?
0
0.1
0.4
0.2
1.3
-1
0
0

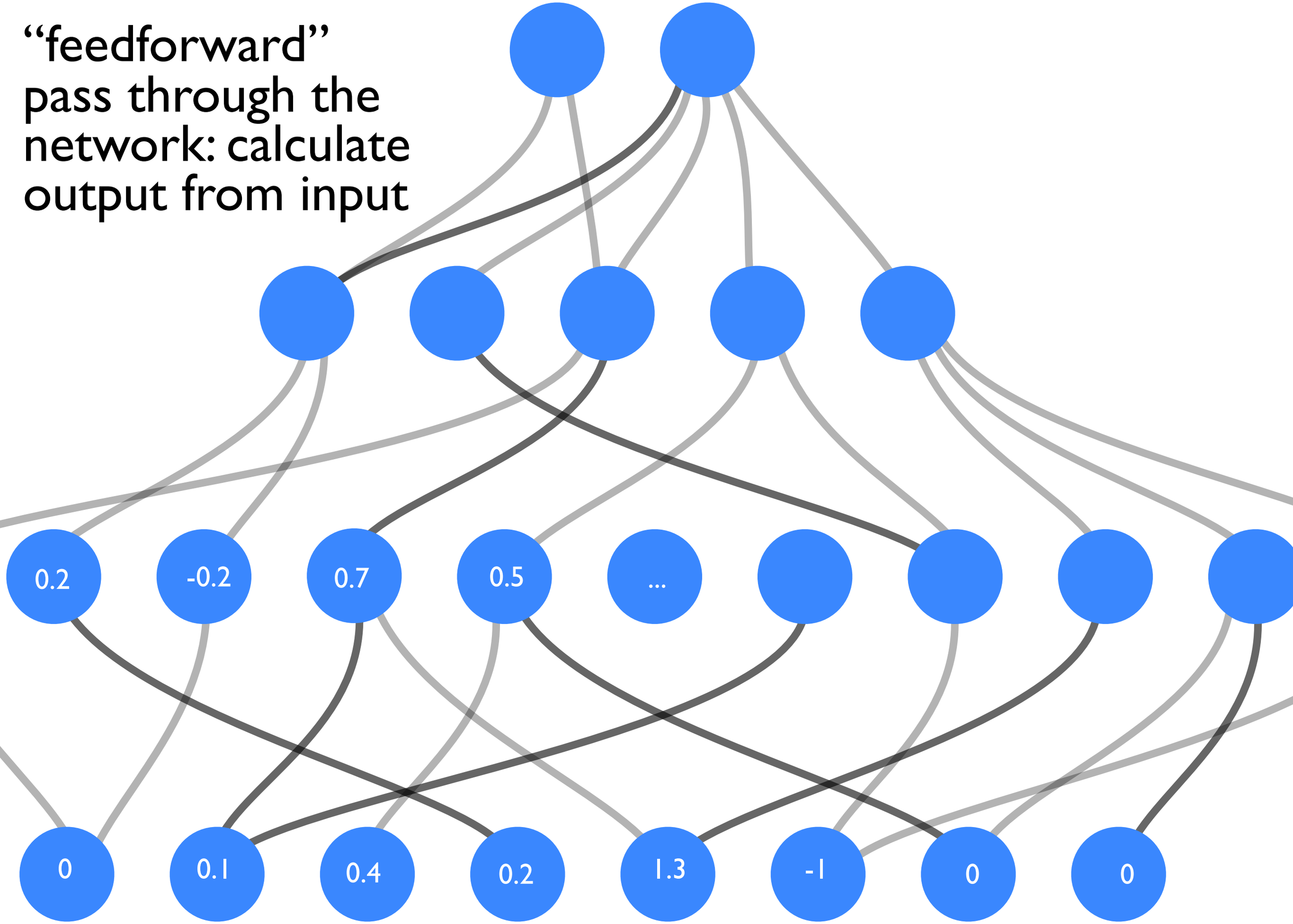

?
0.5
0.1
0
0.1
0.4
0.2
1.3
-1
0
0


f(z)
z=0.5*0.1+0.1*1.3
0.5
0.1
0
0.1
0.4
0.2
1.3
-1
0
0

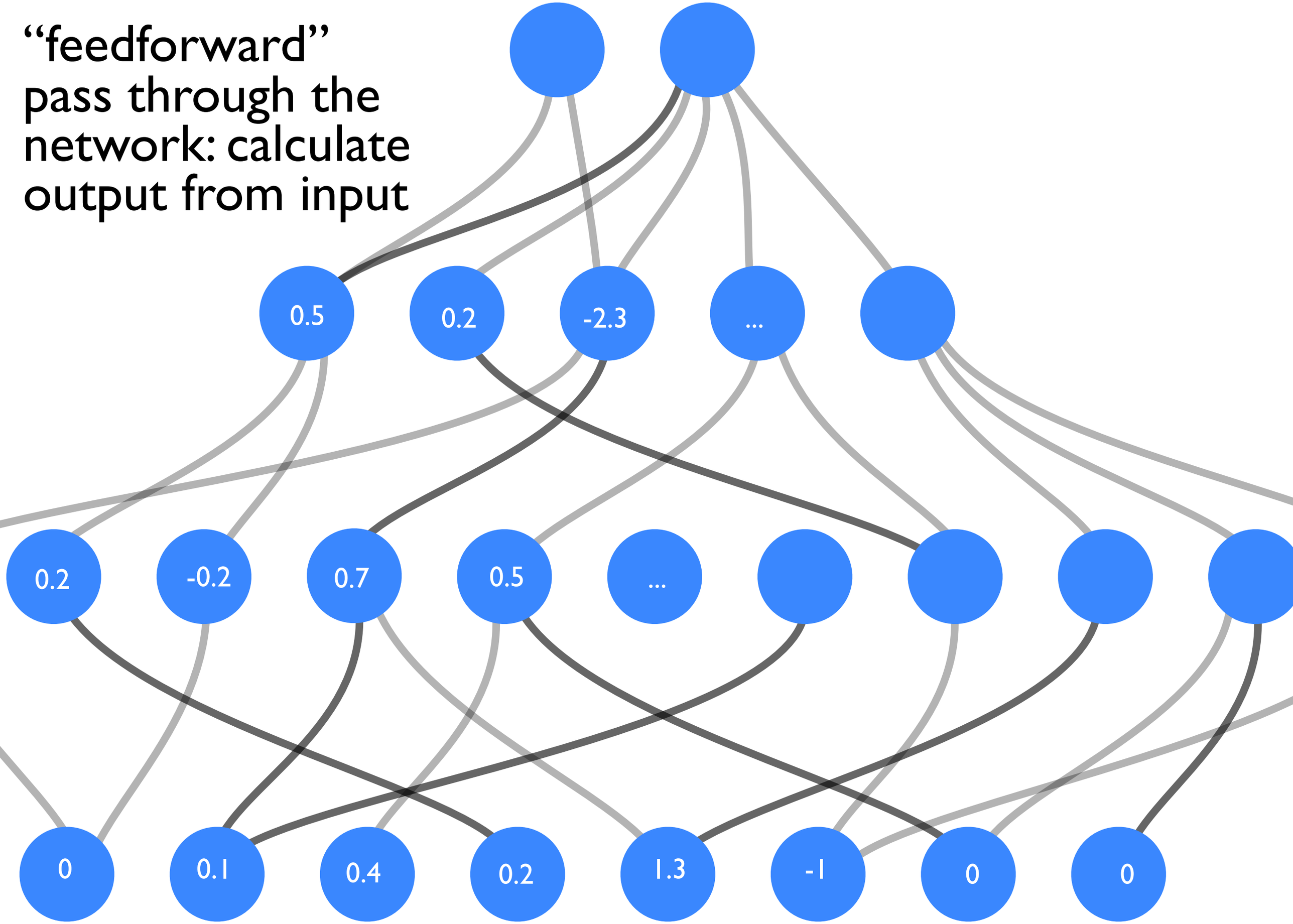
“feedforward” pass through the network: calculate output from input



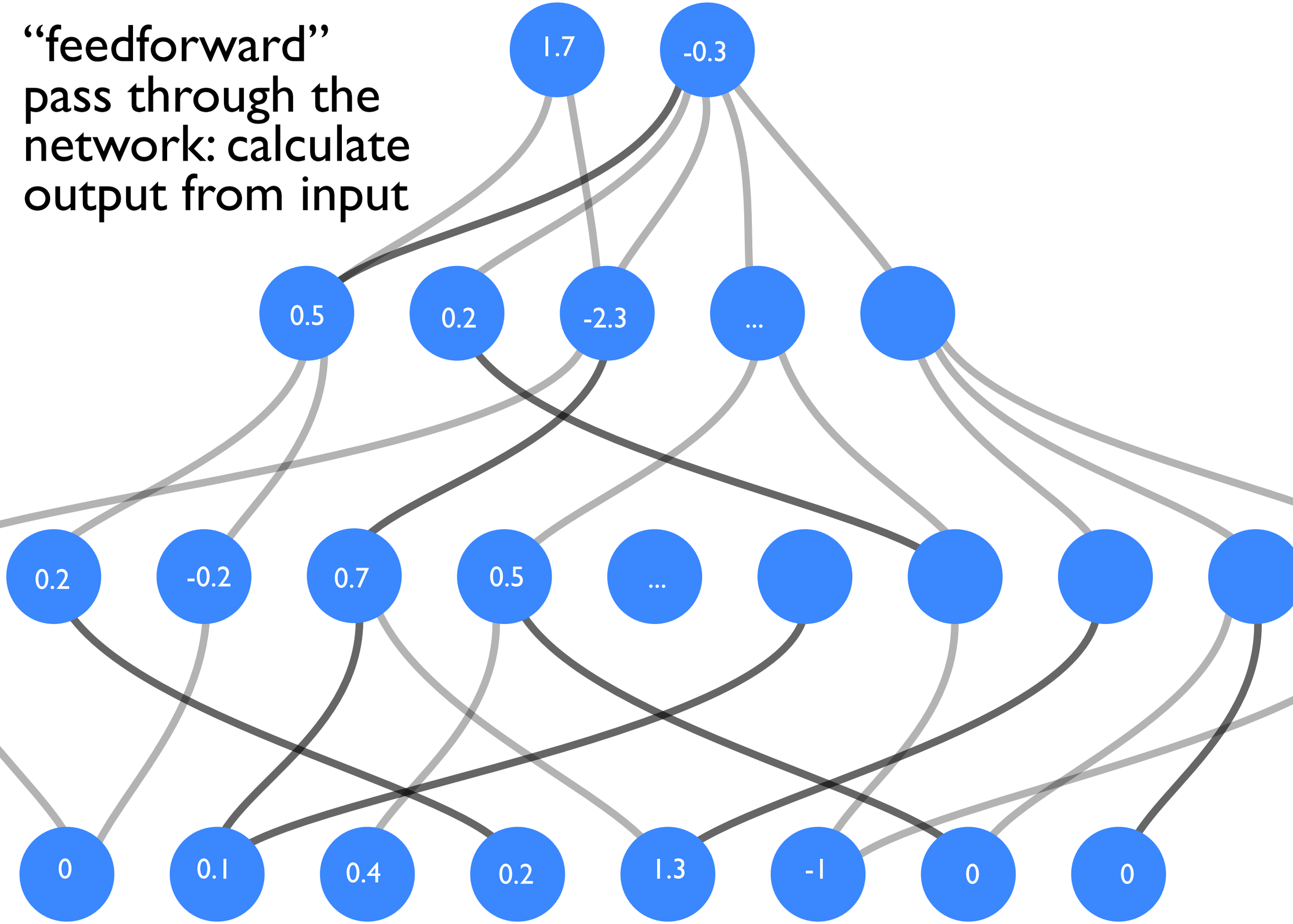
“feedforward” pass through the network: calculate output from input



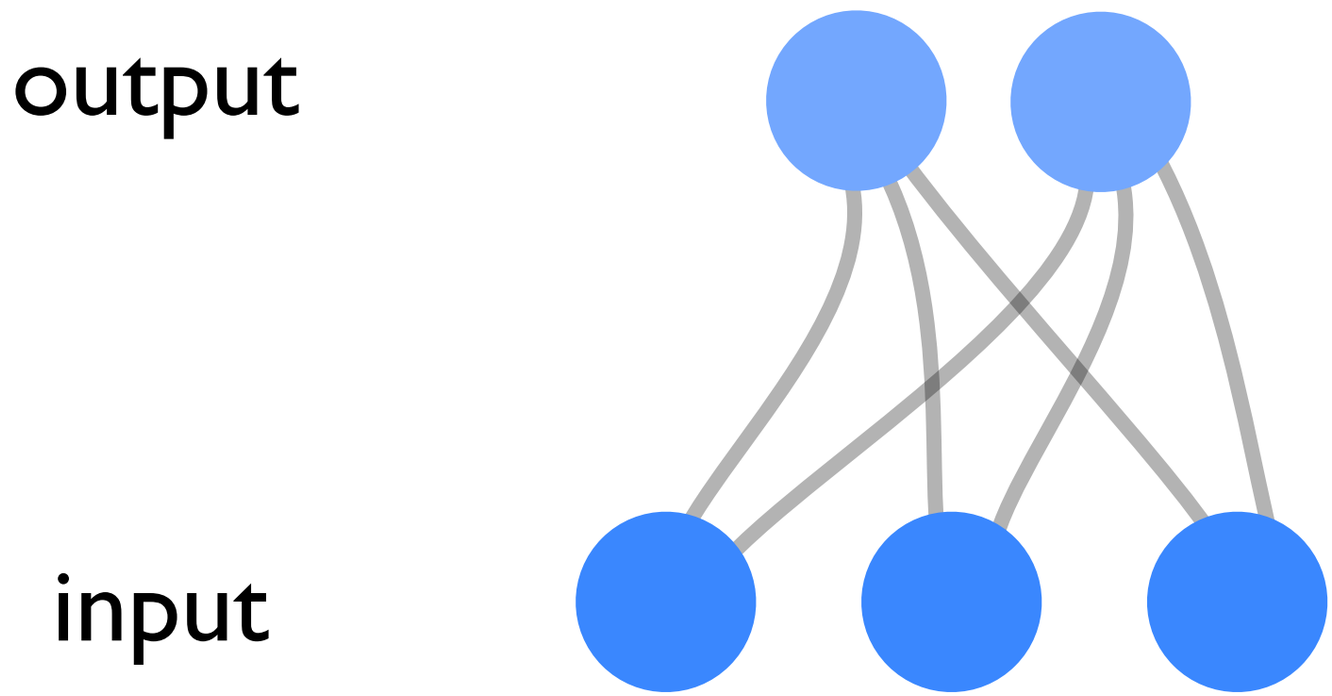
“feedforward” pass through the network: calculate output from input



“feedforward” pass through the network: calculate output from input



One layer



j=output neuron
k=input neuron

$$z_j = \sum_k w_{jk} y_k^{\text{in}} + b_j$$

in matrix/vector notation:

$$z = w y^{\text{in}} + b$$

elementwise nonlinear function:

$$y_j^{\text{out}} = f(z_j)$$

```
for j in range(N):
    do_step(j)
u=dot(M,v)
def f(x):
    return(sin(x)/x)

x=linspace(-5,5,N)
```

Anaconda Jupyter

Python

localhost

AnimTestNew | MachineLearning_Basics | ipython - matplotlib python inline o... | 2017 Machine Learning for Physici...

jupyter MachineLearning_Basics (autosaved)

File Edit View Insert Cell Kernel Help Python [Root]

Code CellToolbar

A very simple neural network (input to output)

In [494]:
```
from numpy import * # get the "numpy" library for linear algebra
```

In [495]:
```
N0=3 # input layer size
N1=2 # output layer size

w=random.uniform(low=-1,high=+1,size=(N1,N0)) # random weights: N1xN0
b=random.uniform(low=-1,high=+1,size=N1) # biases: N1 vector
```

In [496]:
```
y_in=array([0.2,0.4,-0.1]) # input values
```

In [498]:
```
z=dot(w,y_in)+b # result: the vector of 'z' values, length N1
```

Anaconda Jupyter

Python

A few lines of "python" !

Python code

vector (N_{in})

vector (N_{out})

```
z=dot(w,y)+b
```

matrix (N_{out} x N_{in})

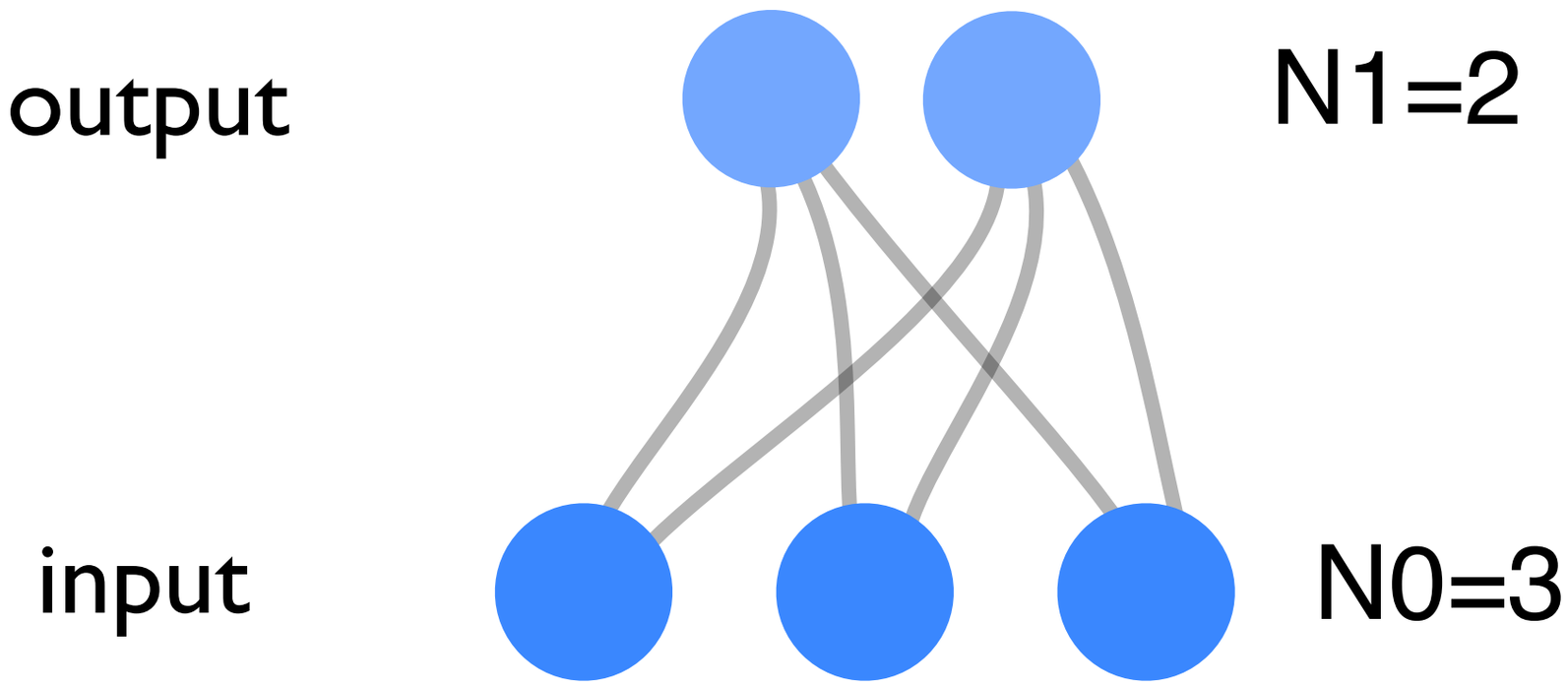
vector (N_{out})

$$z_j = \sum_k w_{jk} y_k^{\text{in}} + b_j$$

in matrix/vector notation:

$$z = w y^{\text{in}} + b$$

A few lines of "python" !



Random weights and biases

```
N0=3 # input layer size
N1=2 # output layer size

w=random.uniform(low=-1,high=+1,size=(N1,N0)) # random weights: N1xN0
b=random.uniform(low=-1,high=+1,size=N1) # biases: N1 vector
```

Input values

```
y_in=array([0.2,0.4,-0.1]) # input values
```

```
z=dot(w,y_in)+b # result: the vector of 'z' values, length N1
y_out=1/(1+exp(-z)) # the sigmoid function (applied elementwise)
```

Apply network!