NSC3270 / NSC5270
Computational Neuroscience

Tu/Th 9:35-10:50am
Featheringill Hall 129

Professor Thomas Palmeri
Professor Sean Polyn

For Today

Required Readings

Chapter 3 (selected pages) of Churchland, P.S., & Sejnowski, T.J. (2017). *The Computational Brain* (25th Anniversary Edition). MIT Press.

In-Class Python Code

Homework3.py

Homework3.ipynb

Info to help with Homework 3

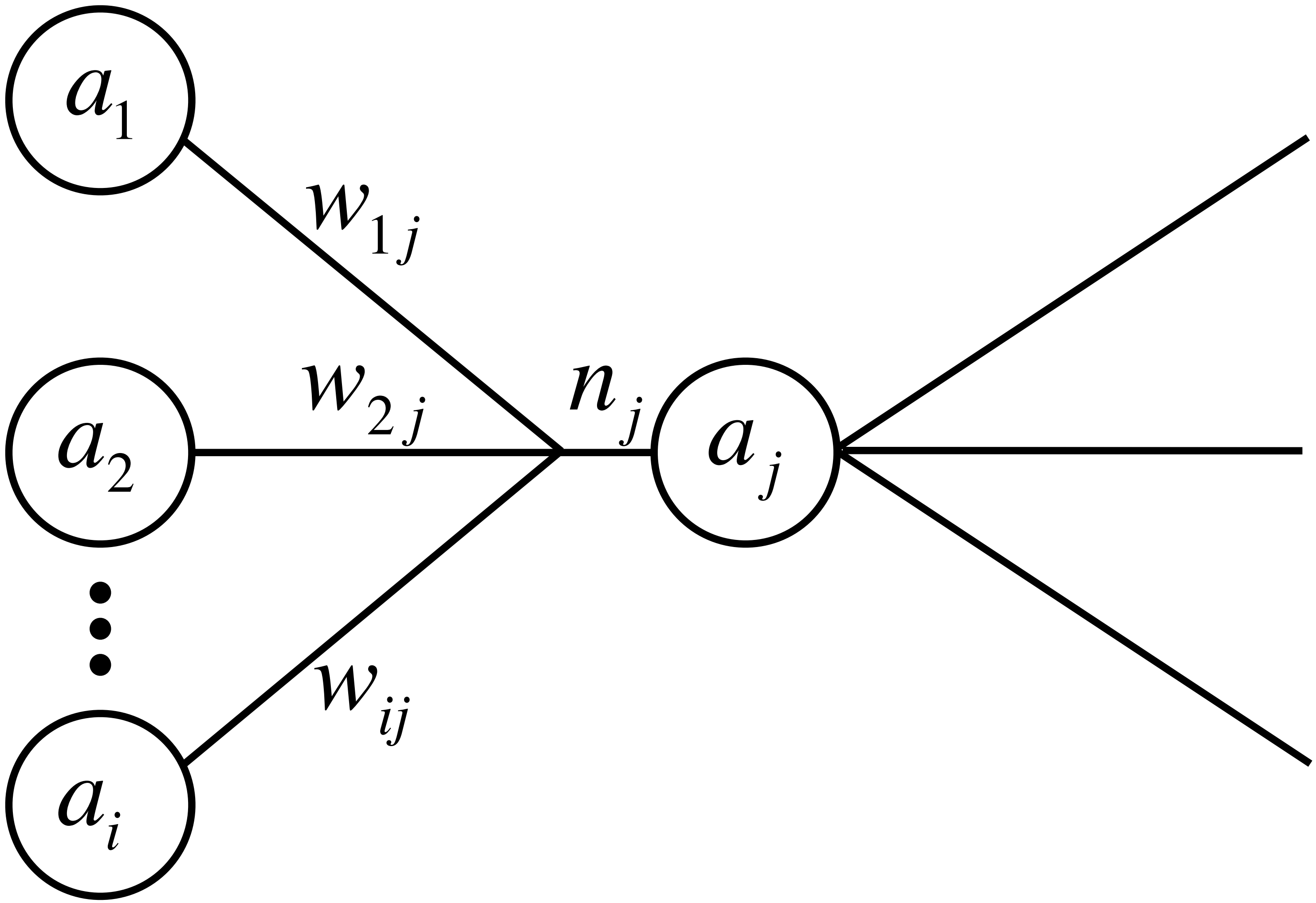
NumpyExamples.ipynb

links on Brightspace

net input sums the weighted inputs

$$n_j = \sum_i a_i w_{ij}$$

inputs integrate at the cell body - they are added together



let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$

$$a = \left[a_1, a_2, \ldots, a_m \right]$$ activation of all the input nodes

$$w_j = \left[w_{1j}, w_{2j}, \ldots, w_{mj} \right]$$ all weights going to 2nd layer node j

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$

$$a = \left[a_1, a_2, \ldots, a_m\right]$$

$$w_j = \left[w_{1j}, w_{2j}, \ldots, w_{mj}\right]$$

```
import numpy as np
n = 0
for i in np.arange(len(a)):
    n += a[i]*wj[i]
```

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$

$$a = \left[a_1, a_2, \ldots, a_m\right]$$

$$w_j = \left[w_{1j}, w_{2j}, \ldots, w_{mj}\right]$$

```
n = sum(a*wj)
```

element-wise multiplication of numpy arrays

(different from Matlab, which requires .* operator)

let's take a look at this equation

$$n_j = \sum_i a_i w_{ij}$$

$$a = \left[a_1, a_2, \ldots, a_m\right]$$

$$w_j = \left[w_{1j}, w_{2j}, \ldots, w_{mj}\right]$$

$$n_j = a \cdot w_j$$

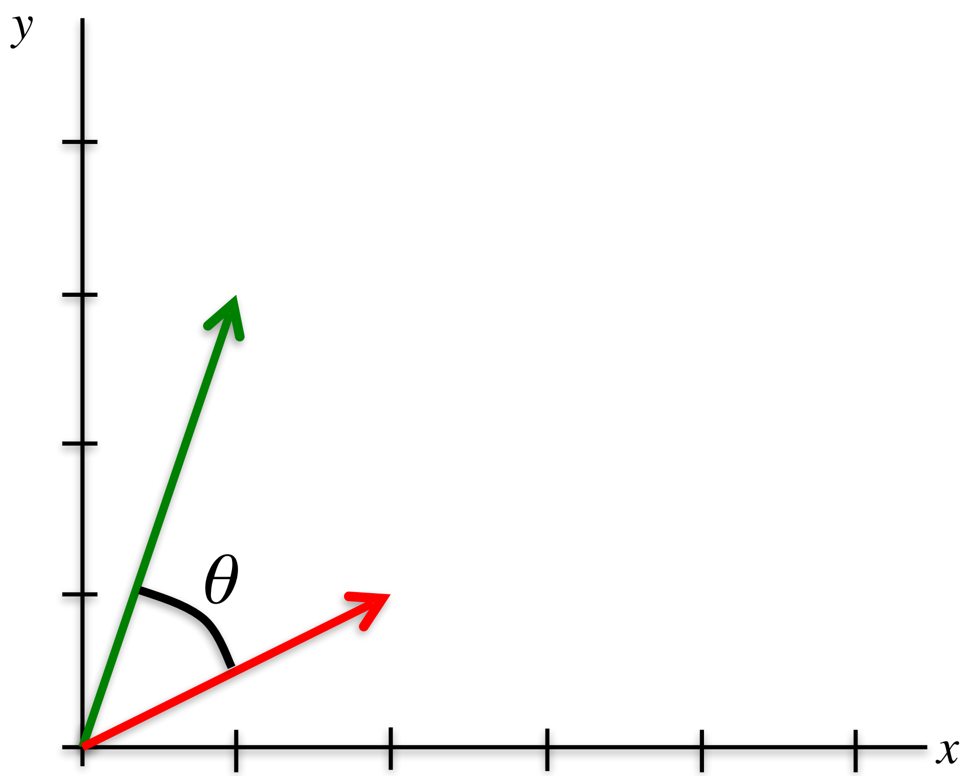
```
import numpy as np
n = np.dot(a,wj)
```

dot product

dot product

"similarity" between two vectors

```
>>> a  = [1, 3]
>>> wj = [2, 1]
```



dot product

angle between two vectors ...

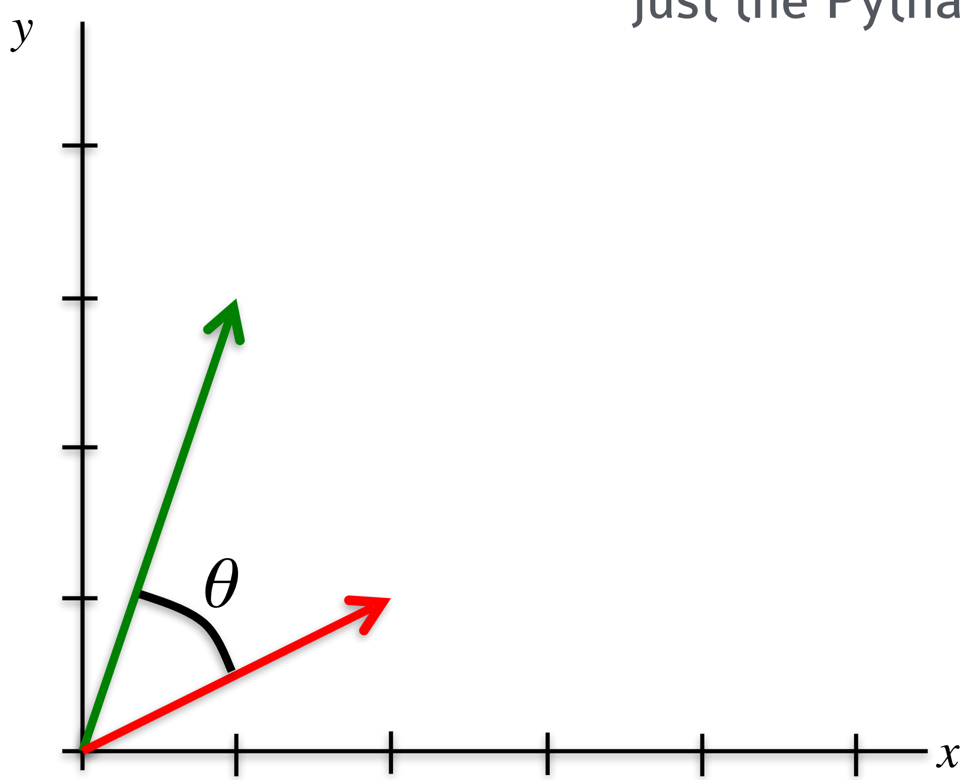
$$\cos(\theta) = \frac{a \cdot w_j}{\| a \| \| w_j \|}$$

dot product

norm

$$\| a \| = \sqrt{a \cdot a}$$

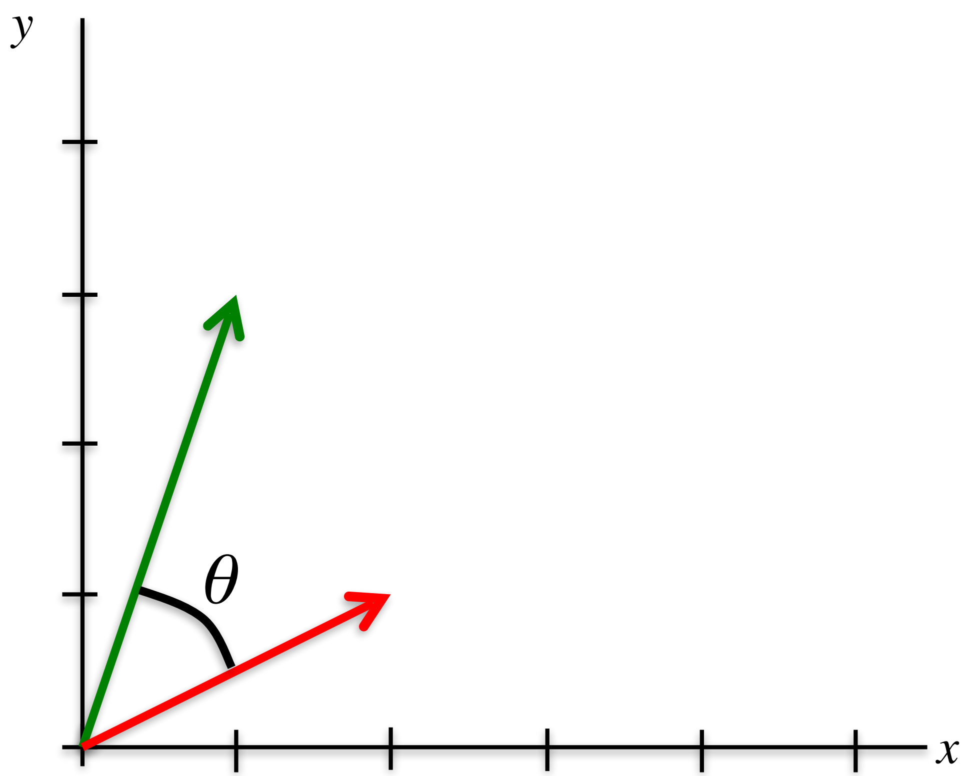
just the Pythagorean theorem



dot product

```
>>> theta = m.degrees(m.acos(np.dot(a,wj) /
    (np.linalg.norm(a)*np.linalg.norm(wj))))
```

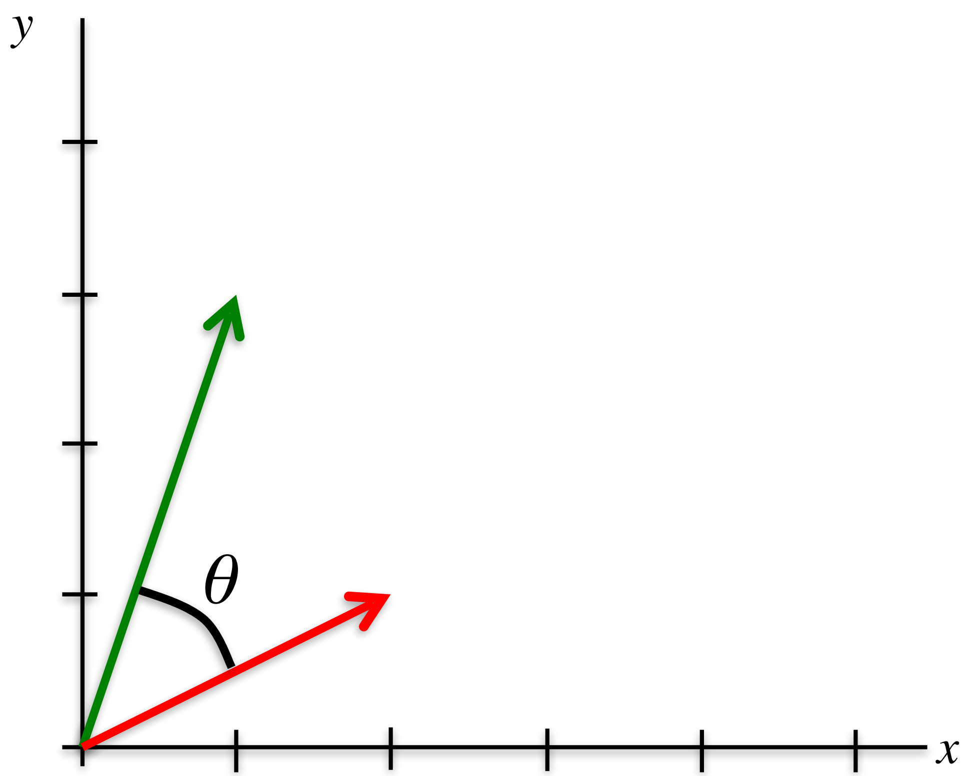
$$\cos(\theta) = \frac{a \cdot w_j}{\| a \| \| w_j \|}$$



dot product

```
>>> theta = m.degrees(m.acos(np.dot(a,wj) /
      (np.dot(a,a)*np.dot(wj,wj))))
```

$$\cos(\theta) = \frac{a \cdot w_j}{\| a \| \| w_j \|}$$

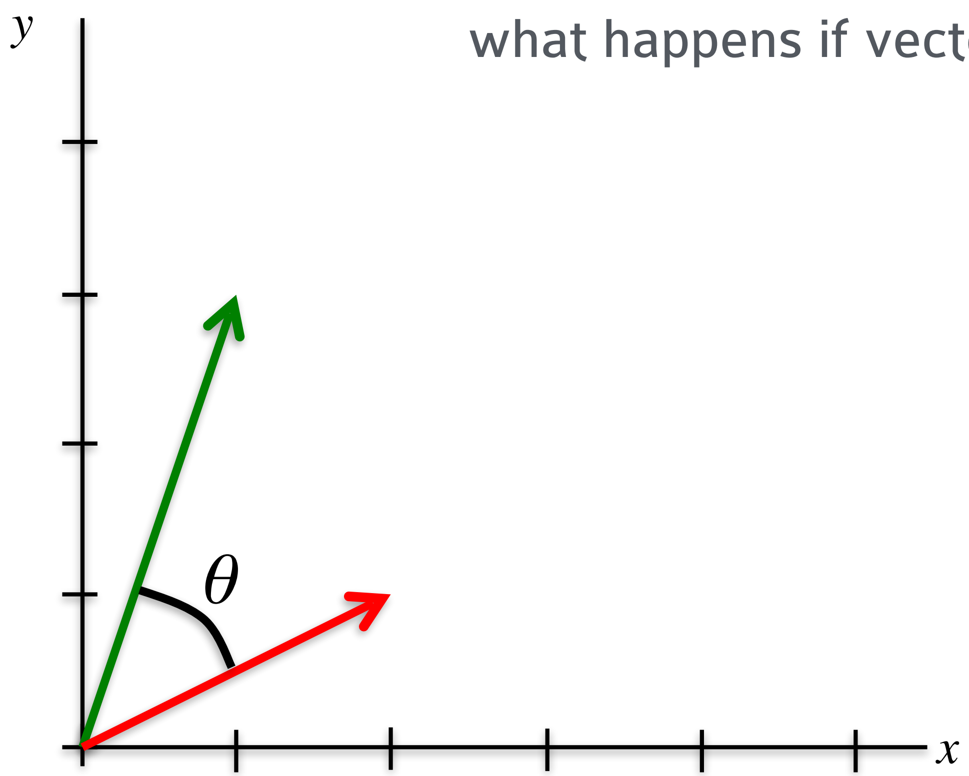


dot product

product of the lengths and the angle

$$a \cdot w_j = \| a \| \| w_j \| \cos(\theta)$$

dot product	length a	length w_j	cos angle



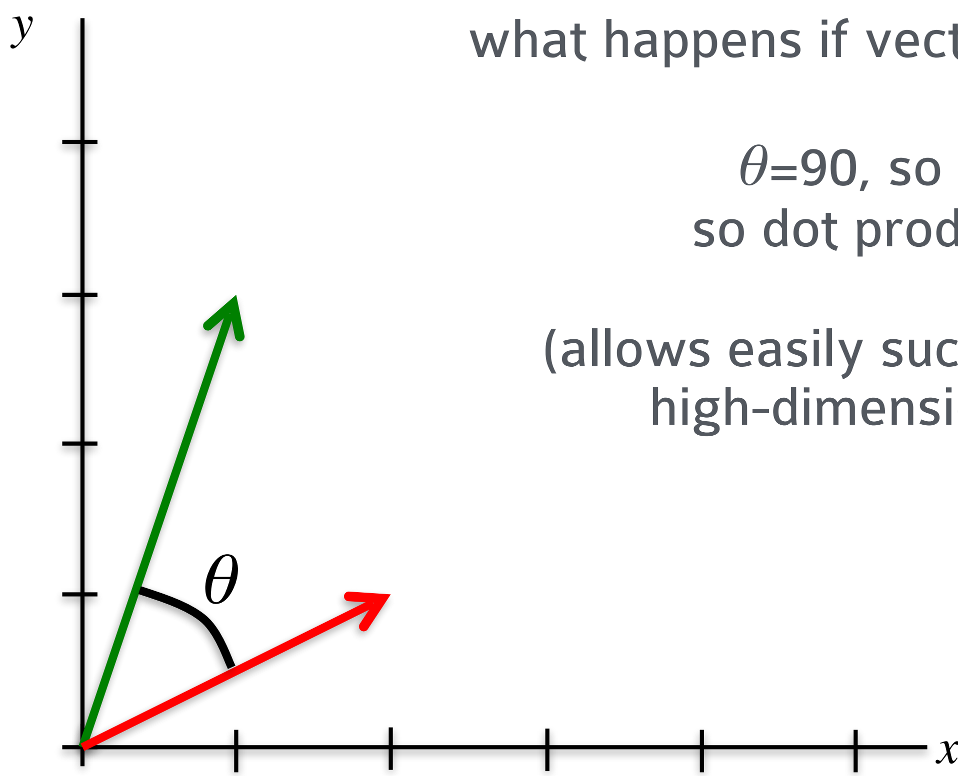
what happens if vectors are orthogonal?

dot product

product of the lengths and the angle

$$a \cdot w_j = \| a \| \| w_j \| \cos(\theta)$$

dot product	length a	length w_j	cos angle

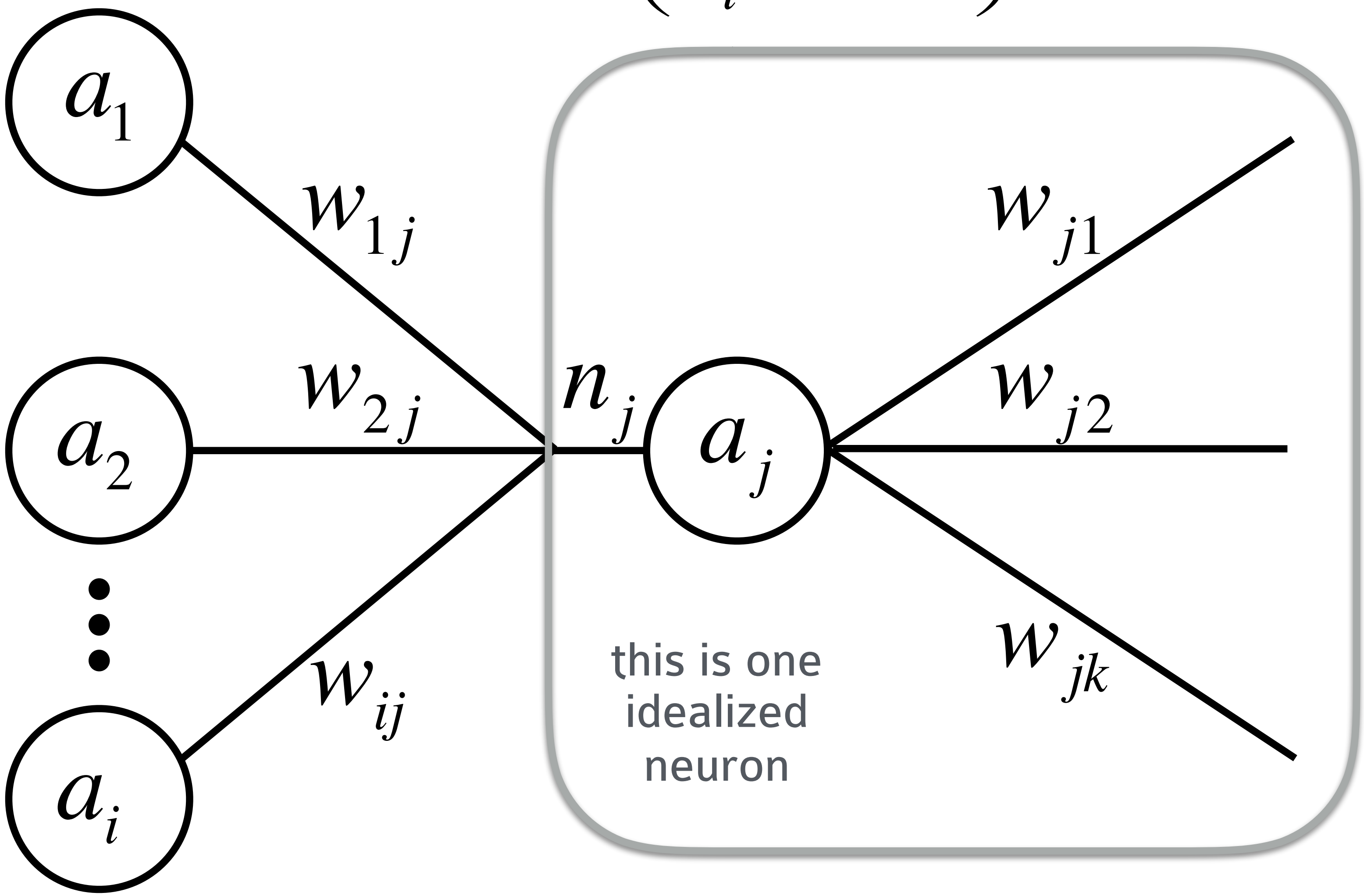


what happens if vectors are orthogonal?

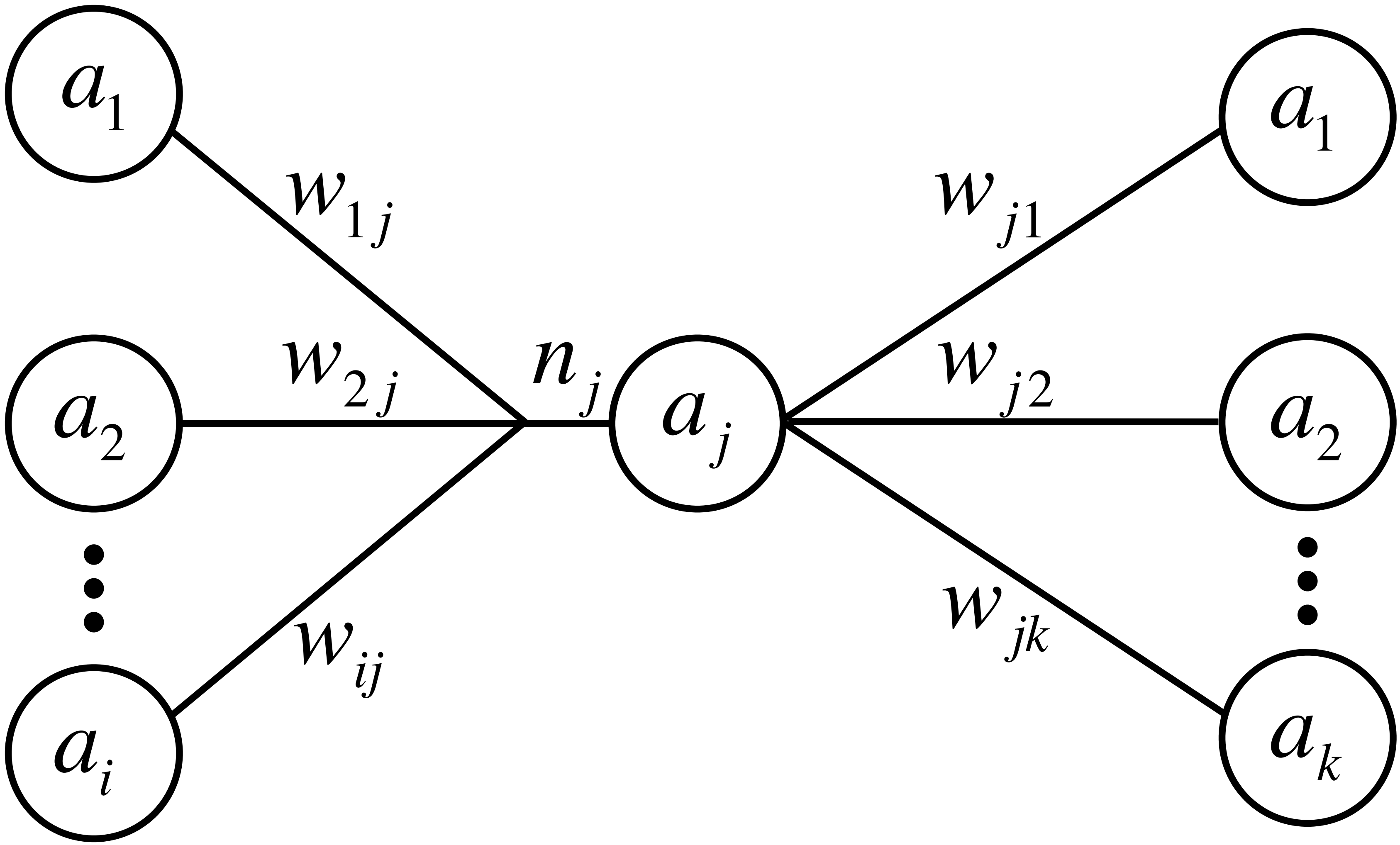
θ=90, so cos(θ)=0
so dot product is zero

(allows easily such computation in high-dimensional spaces) activation equation

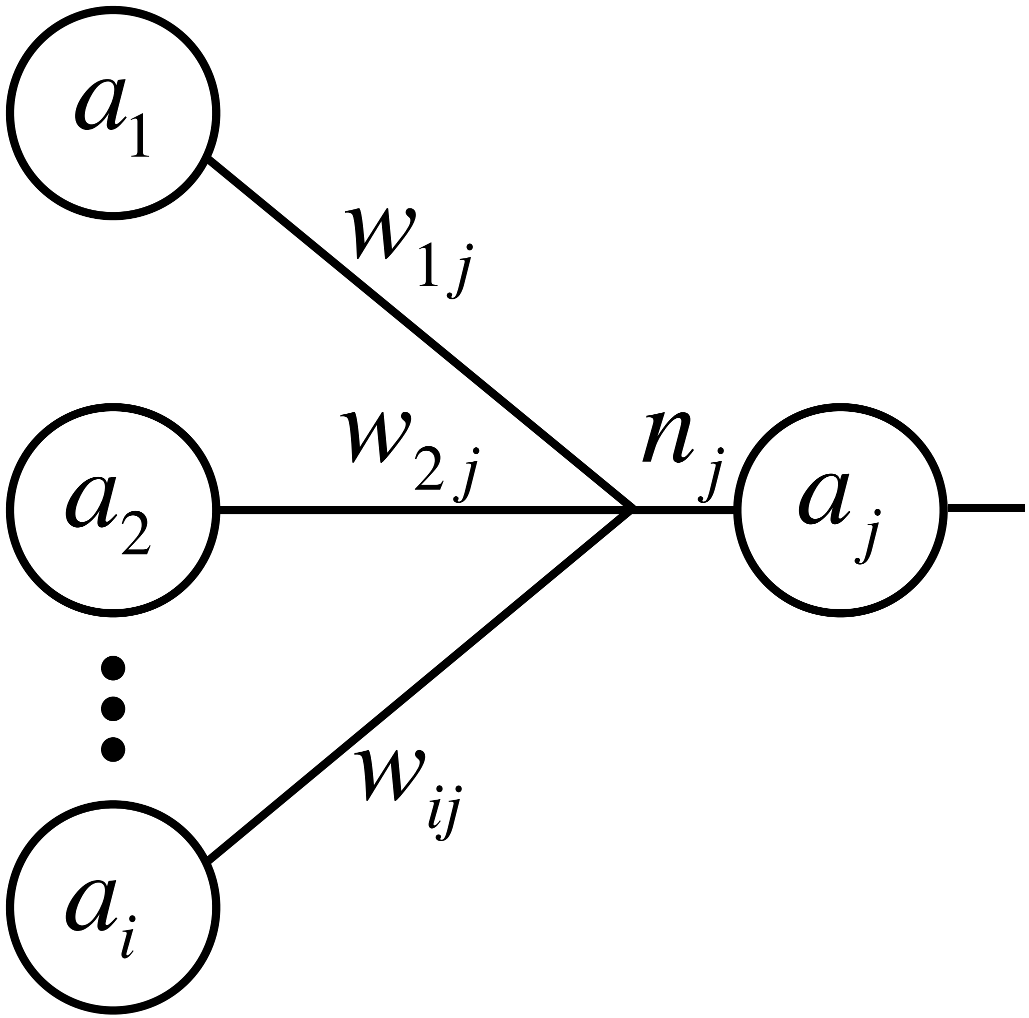
$$a_j = f\left(\sum_i a_i w_{ij}\right)$$



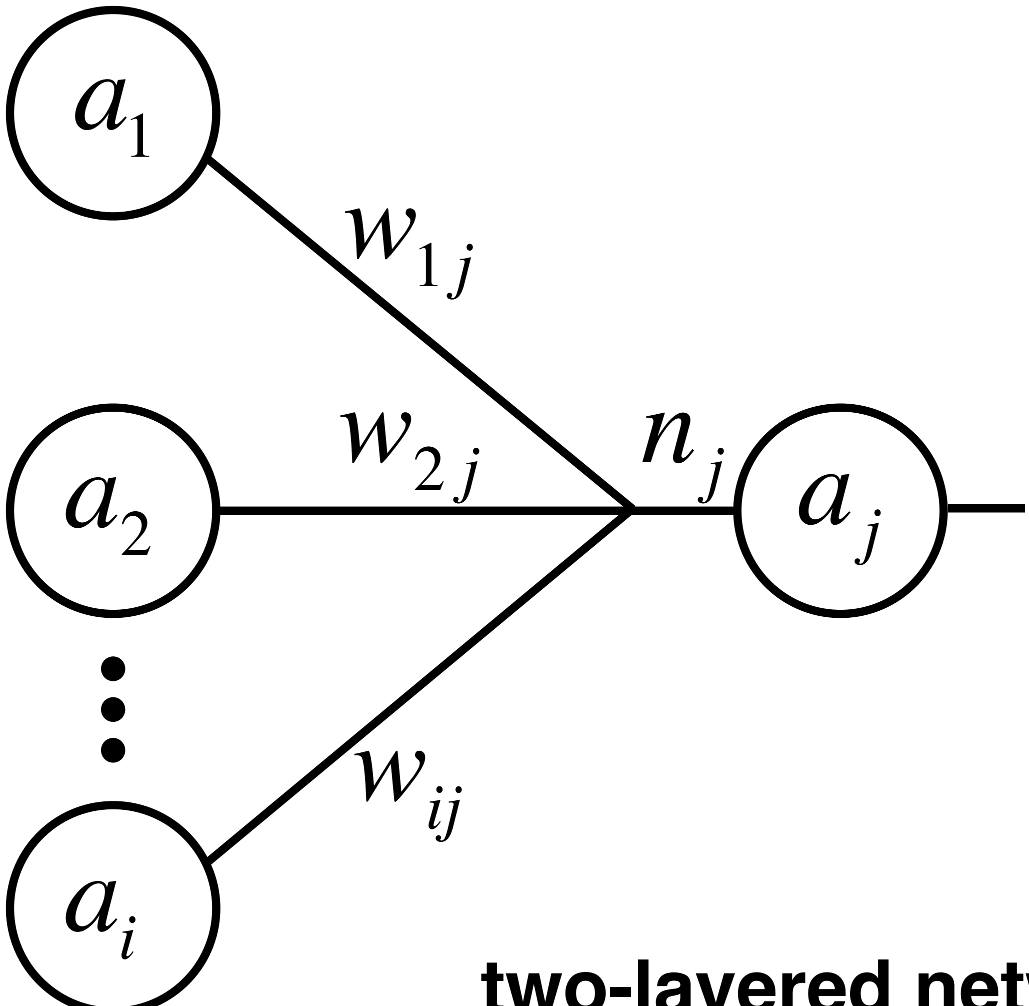
Idealized Neuron



Multi-layered network



single-layered network

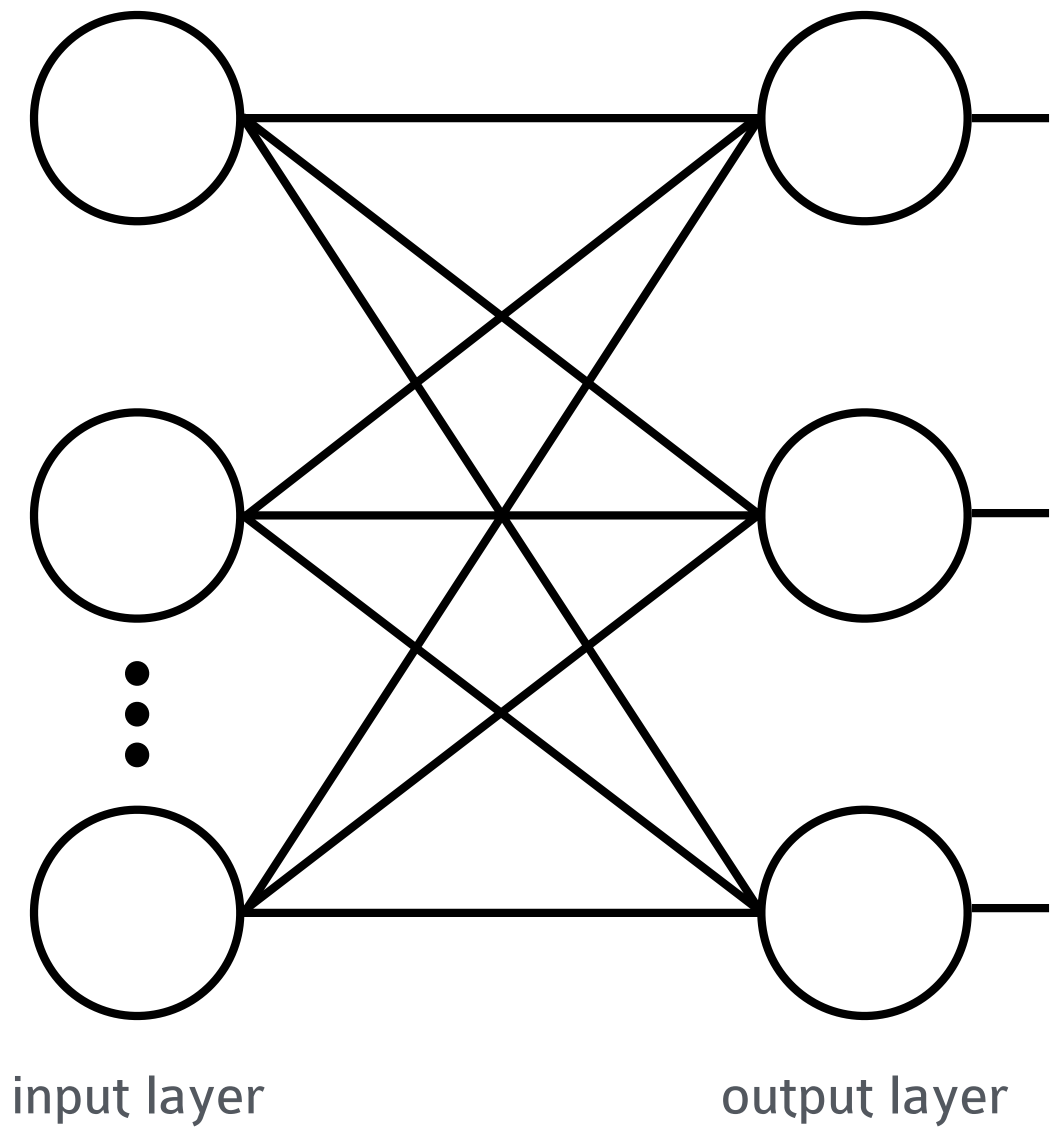


two-layered network

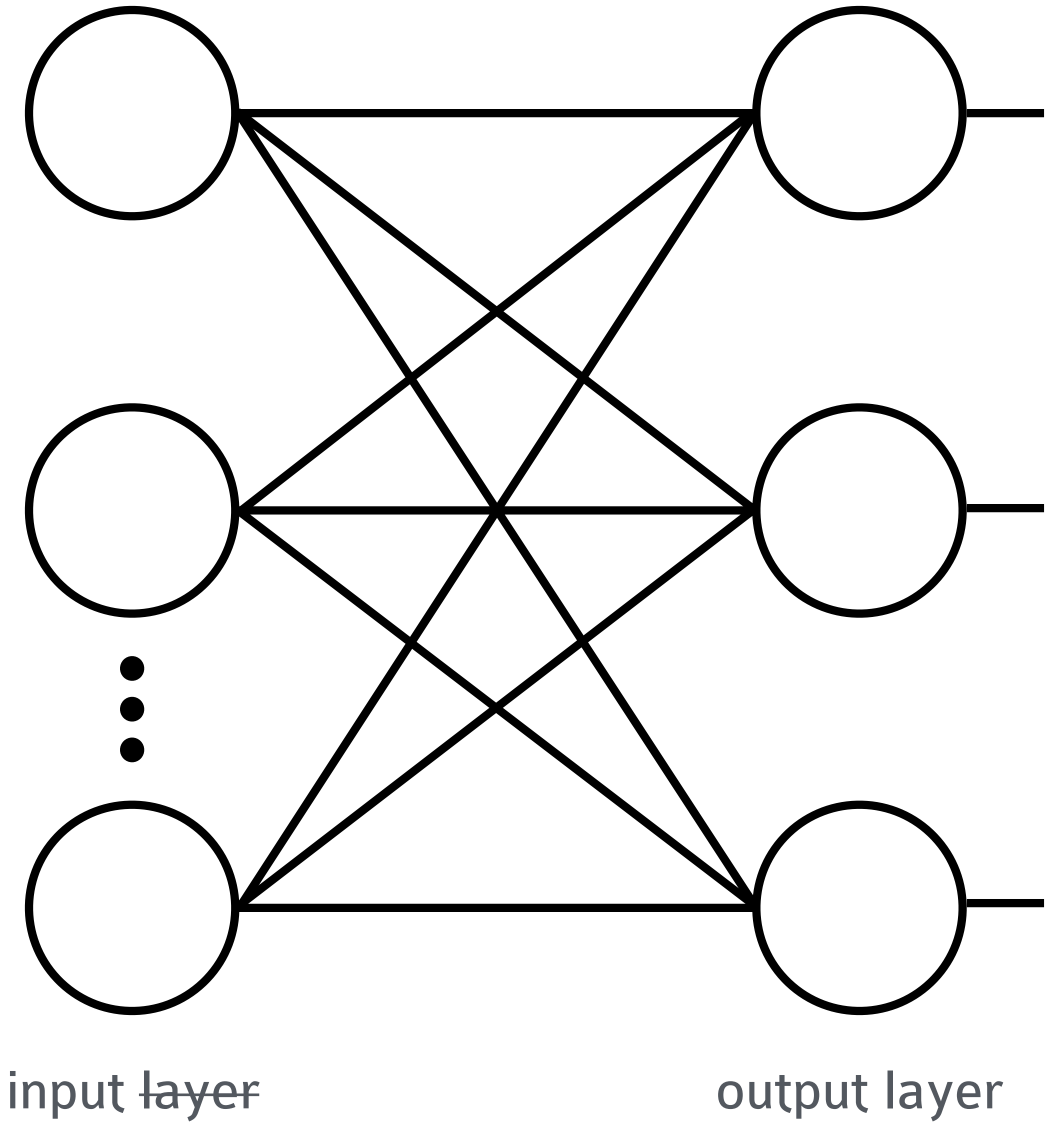
~~single-layered network~~

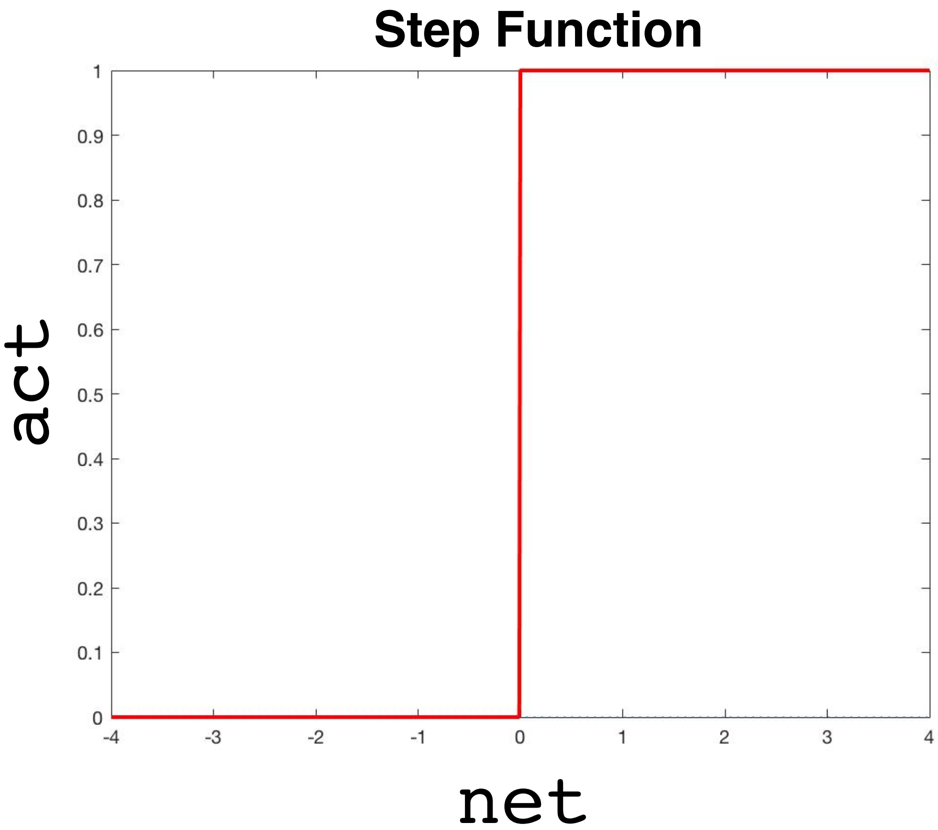
the terminology
may vary

fully-interconnected two-layer network



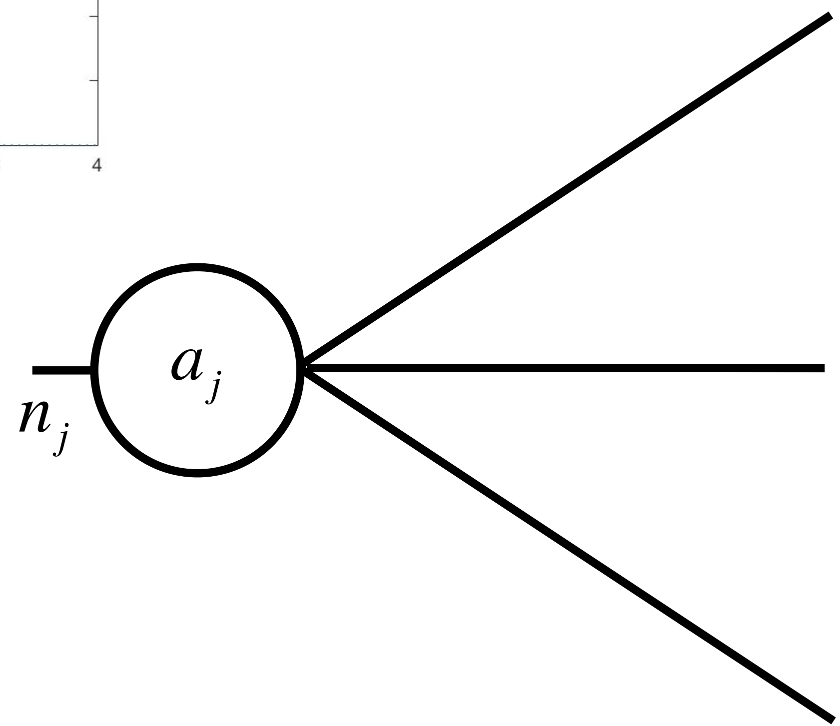
fully-interconnected
~~two-layer network~~
single-layer network

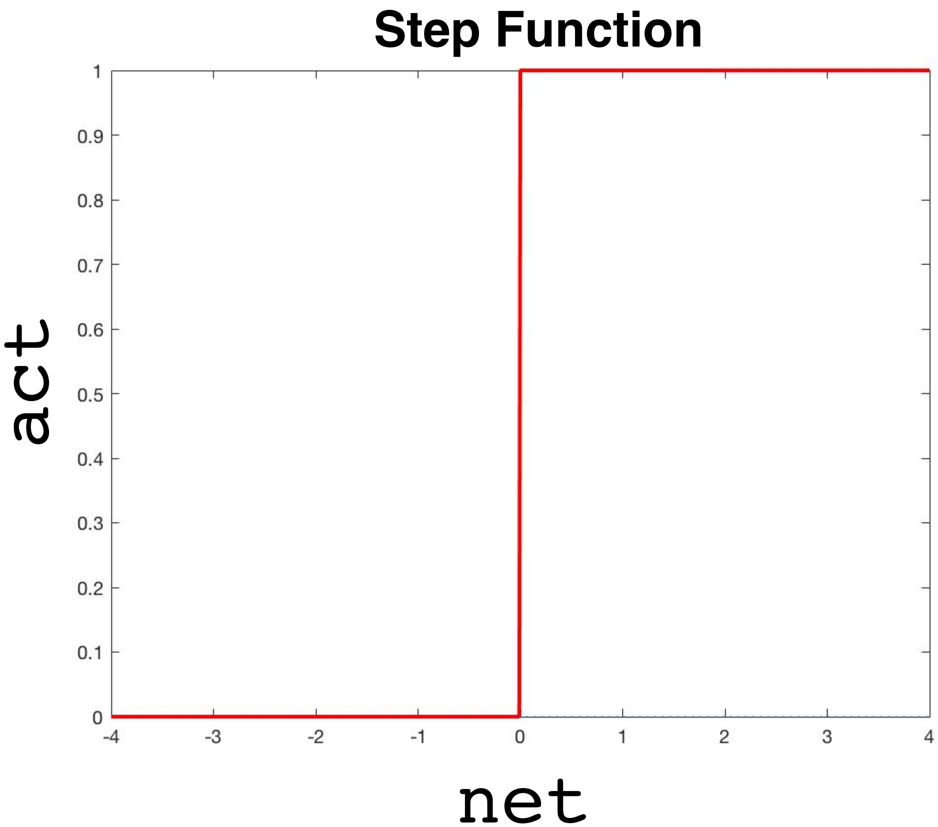




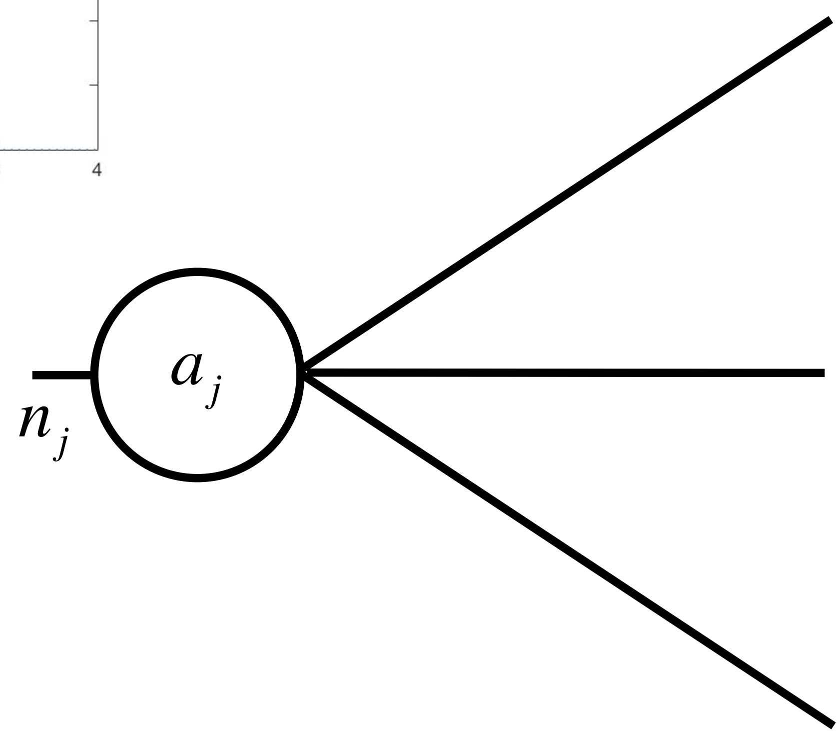
neurons have a nonlinear response function

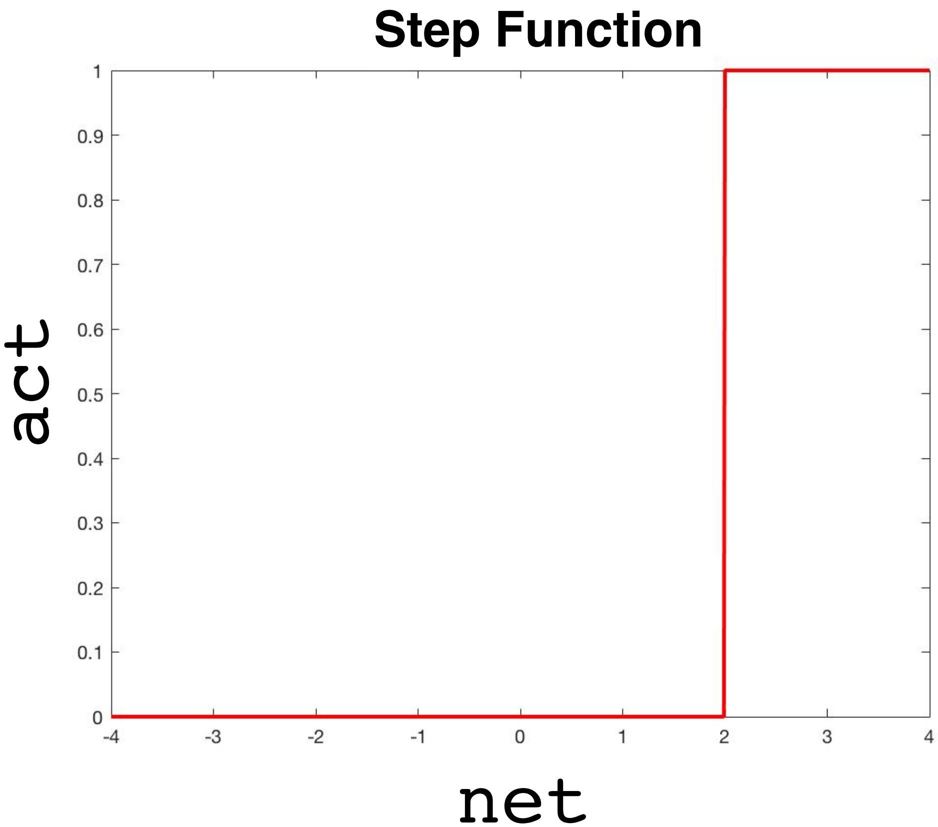
$$a_j = \begin{cases} 0 & \text{if } n_j \leq 0 \\ 1 & \text{otherwise} \end{cases}$$



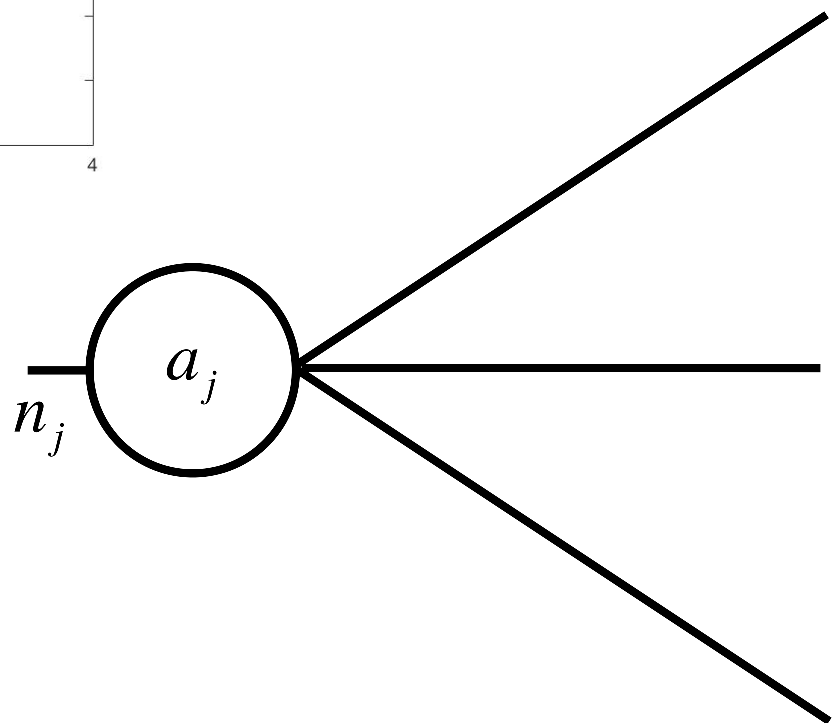


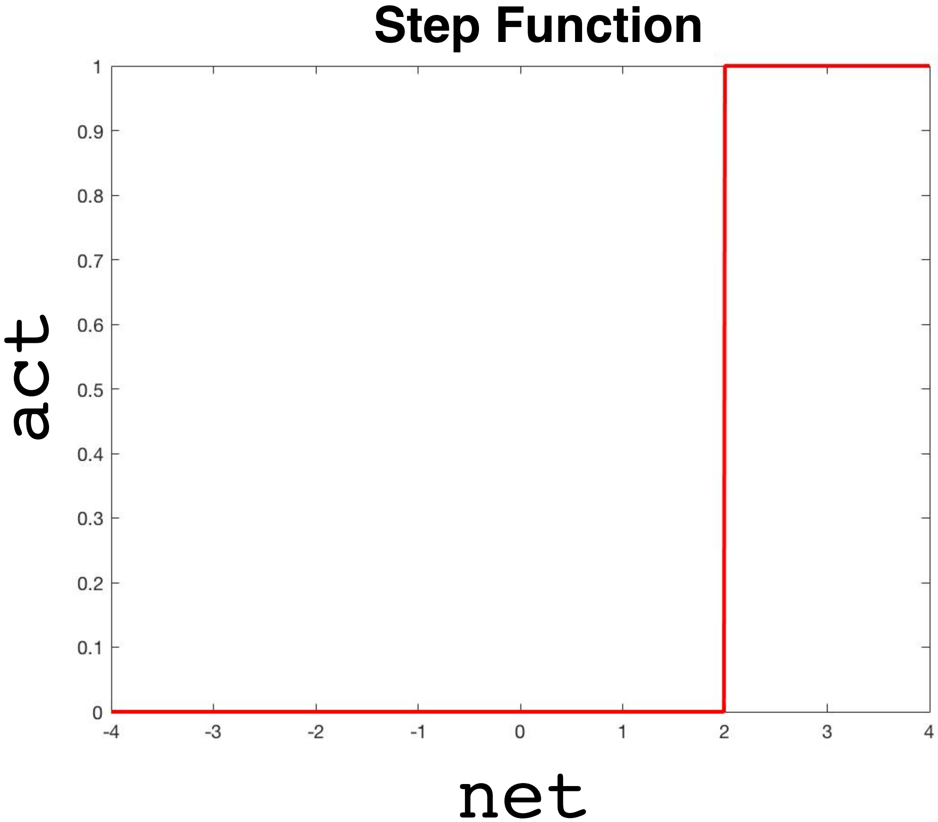
$$a_j = \begin{cases} 0 & \text{if } n_j \le \boxed{0} \\ 1 & \text{otherwise} \end{cases}$$



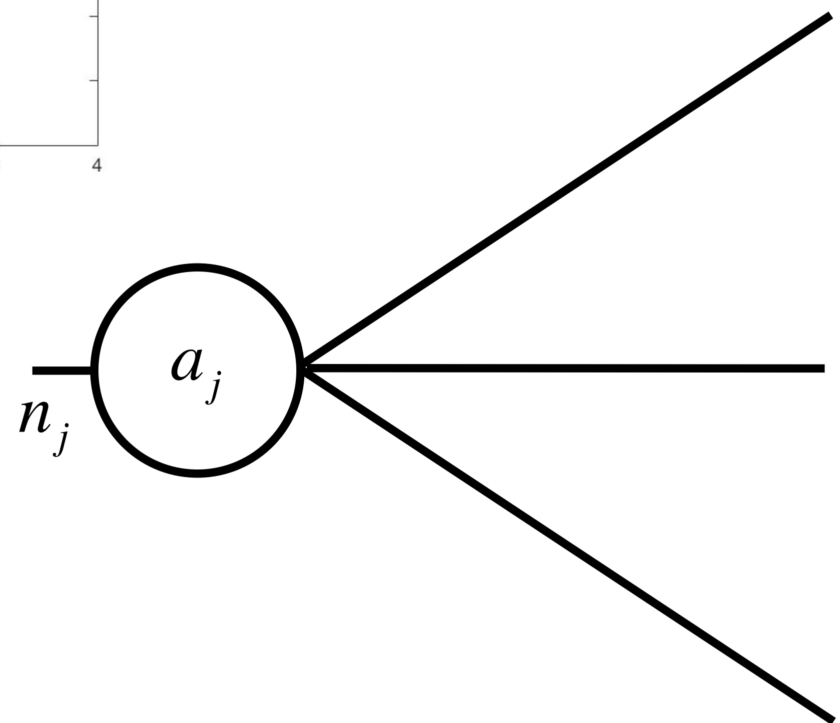


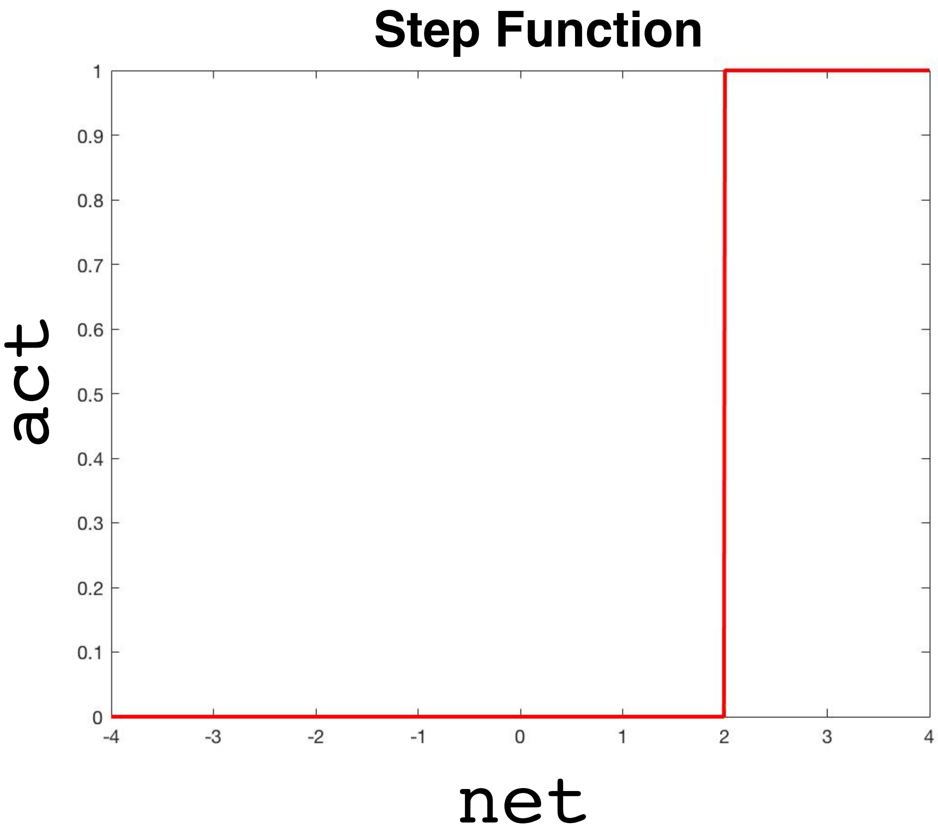
$$a_j = \begin{cases} 0 & \text{if } n_j \le \boxed{2} \\ 1 & \text{otherwise} \end{cases}$$



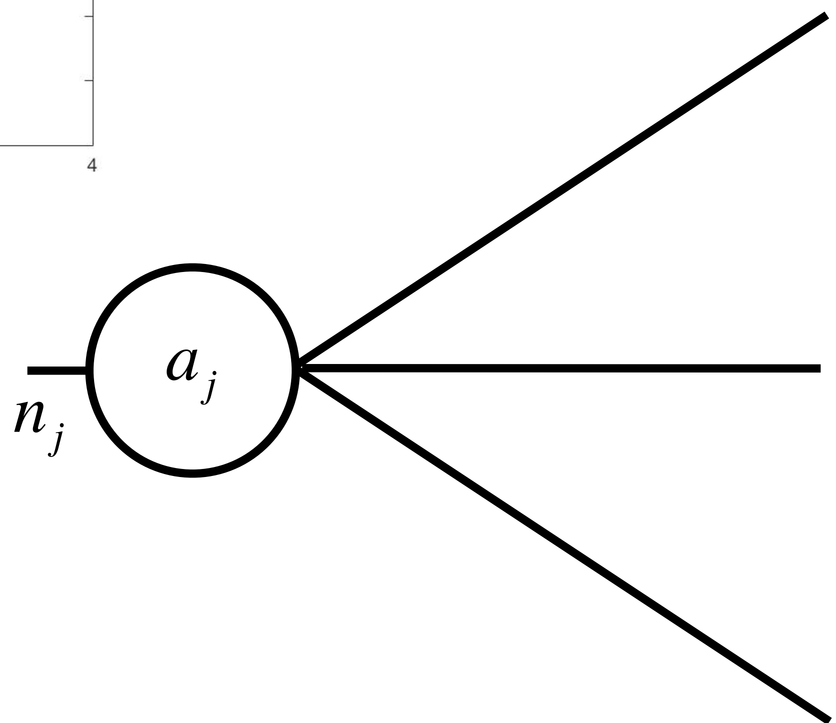


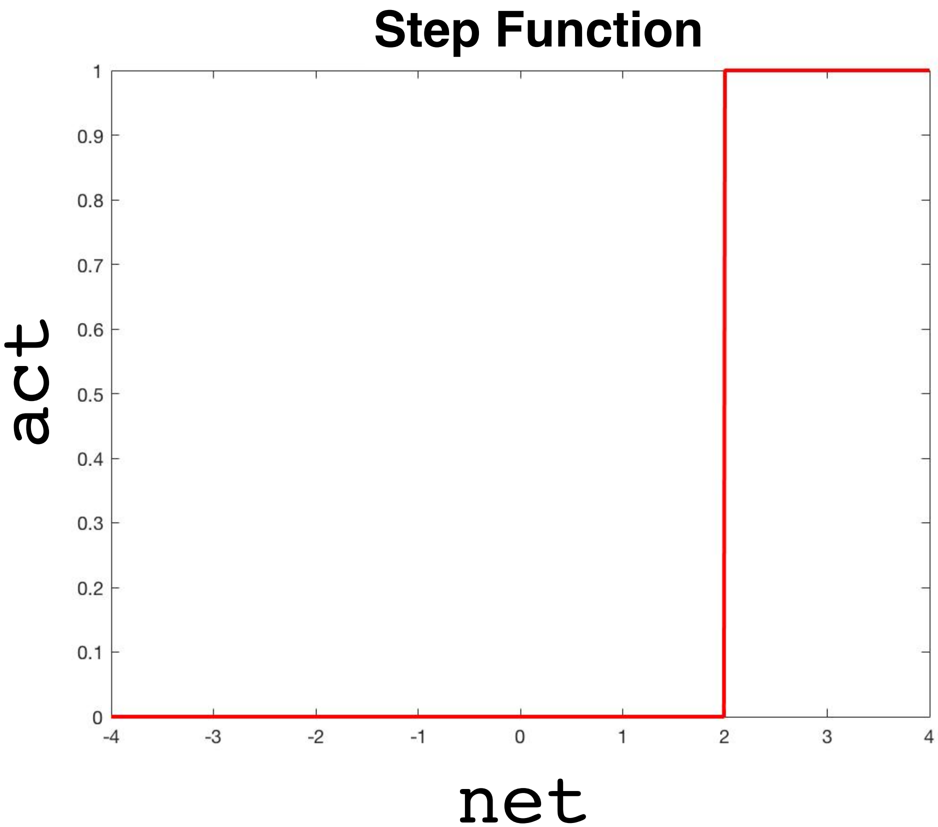
$$a_j = \begin{cases} 0 & \text{if } n_j - 2 \le 0 \\ 1 & \text{otherwise} \end{cases}$$



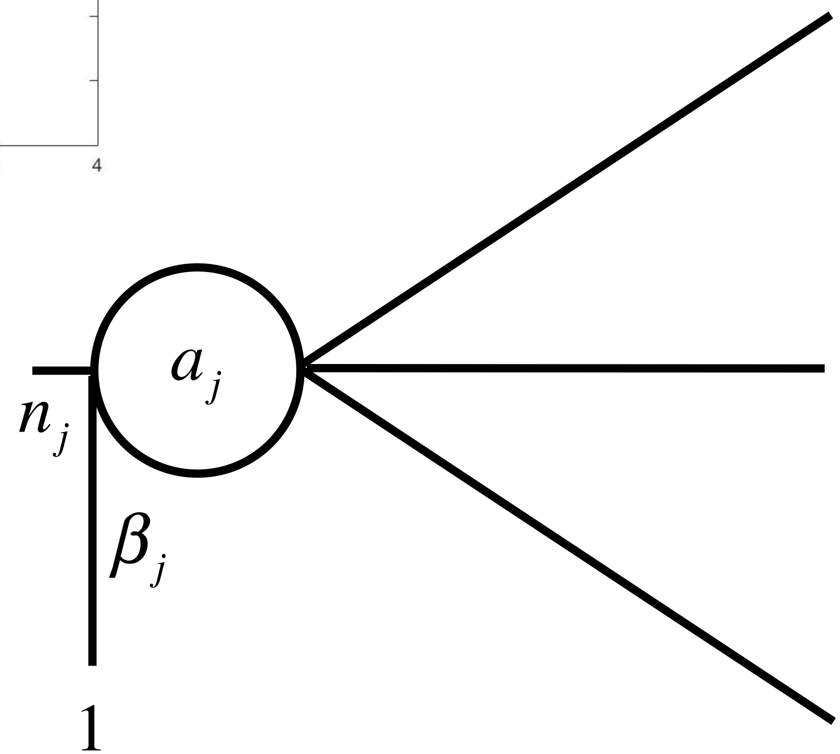


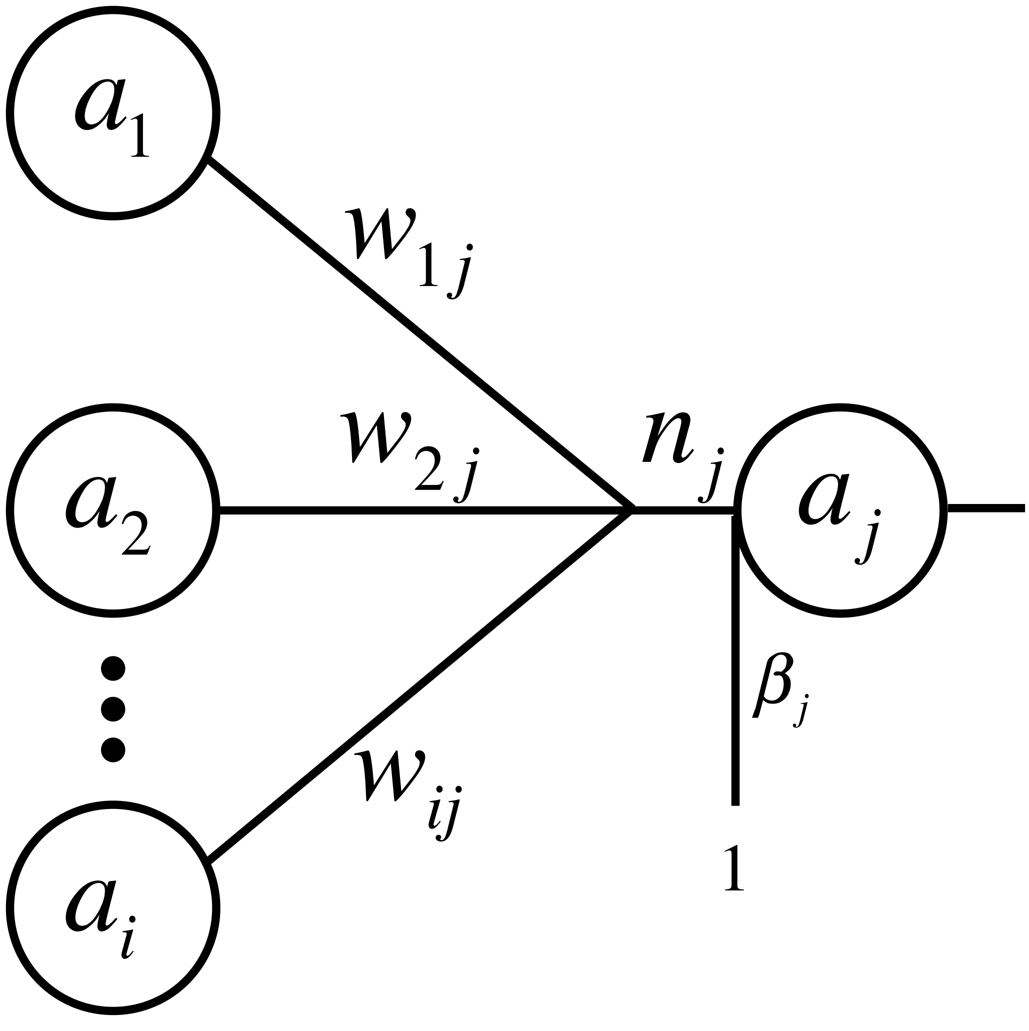
$$a_j = \begin{cases} 0 & \text{if } n_j + (-2) \times 1 \leq 0 \\ 1 & \text{otherwise} \end{cases}$$



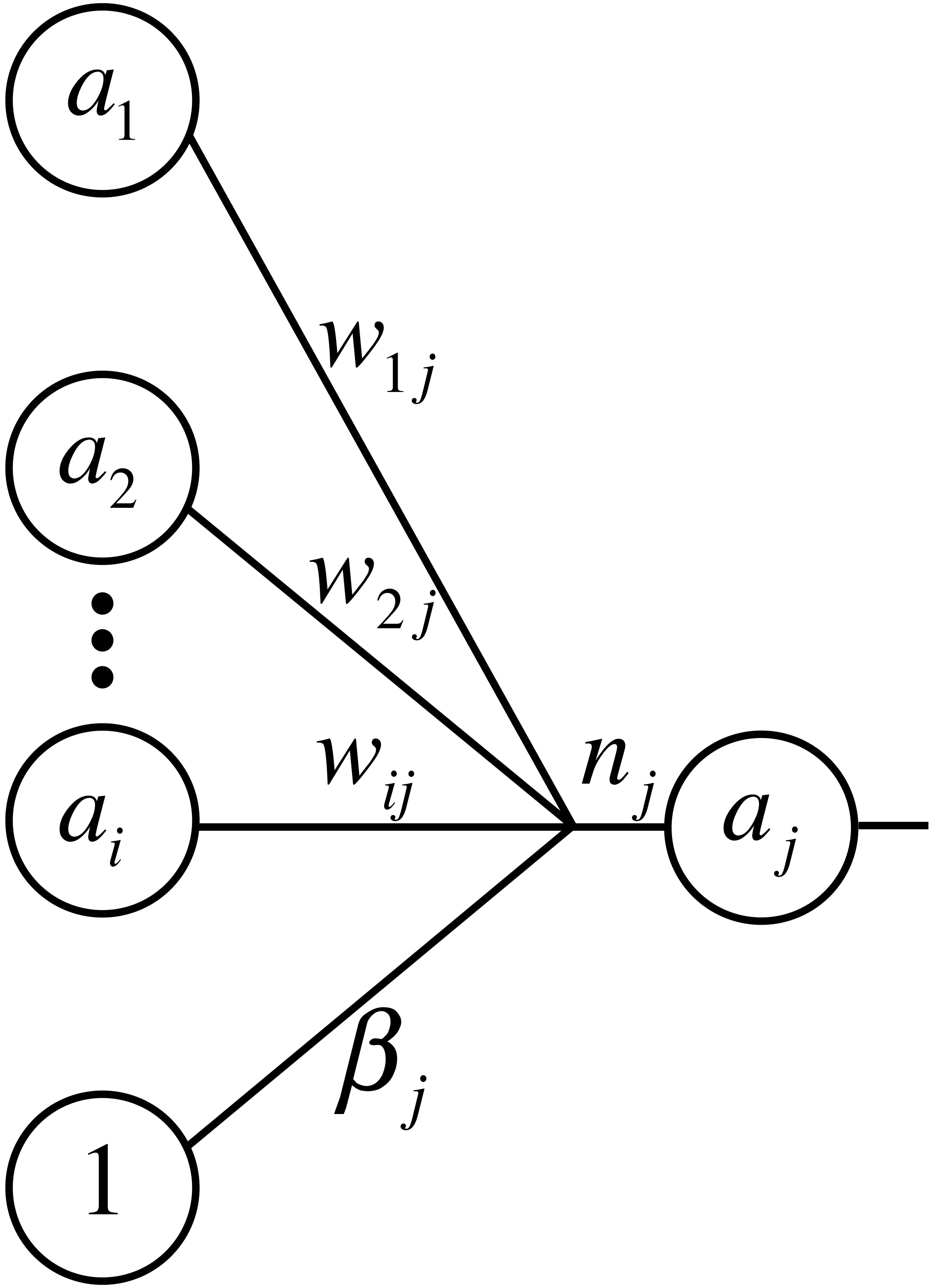


$$a_j = \begin{cases} 0 & \text{if } n_j + (\beta_j) \times 1 \le 0 \\ 1 & \text{otherwise} \end{cases}$$

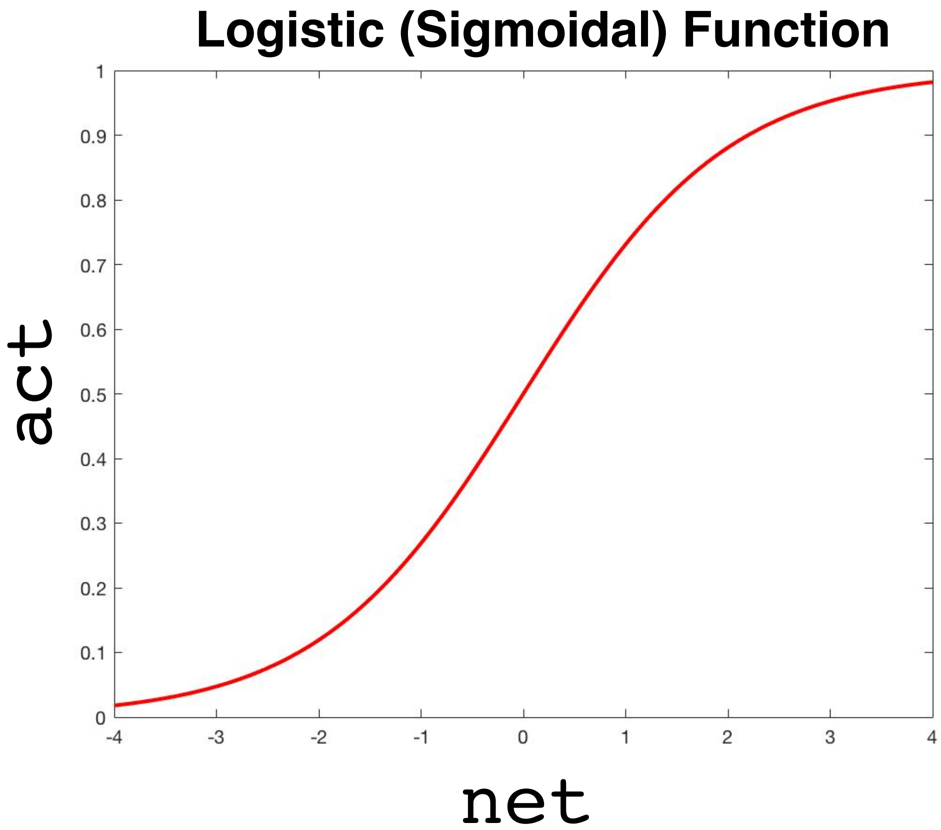


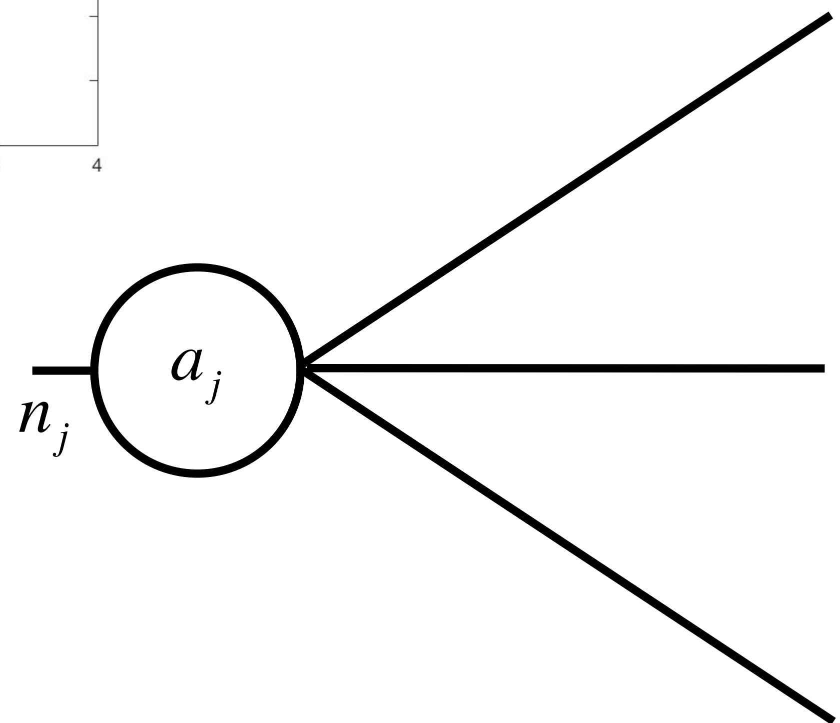


a_1
a_2
a_i
w_{1j}
w_{2j}
w_{ij}
n_j
a_j
β_j
1



a_1
w_{1j}
a_2
w_{2j}
w_{ij}
n_j
a_i
a_j
β_j
1

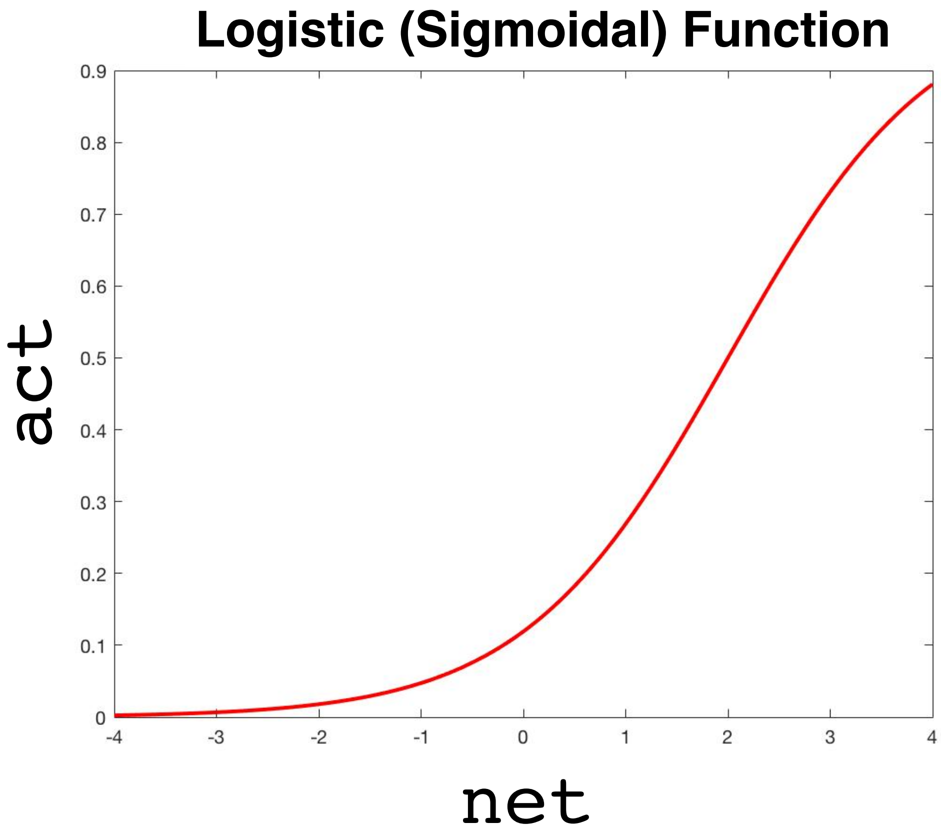




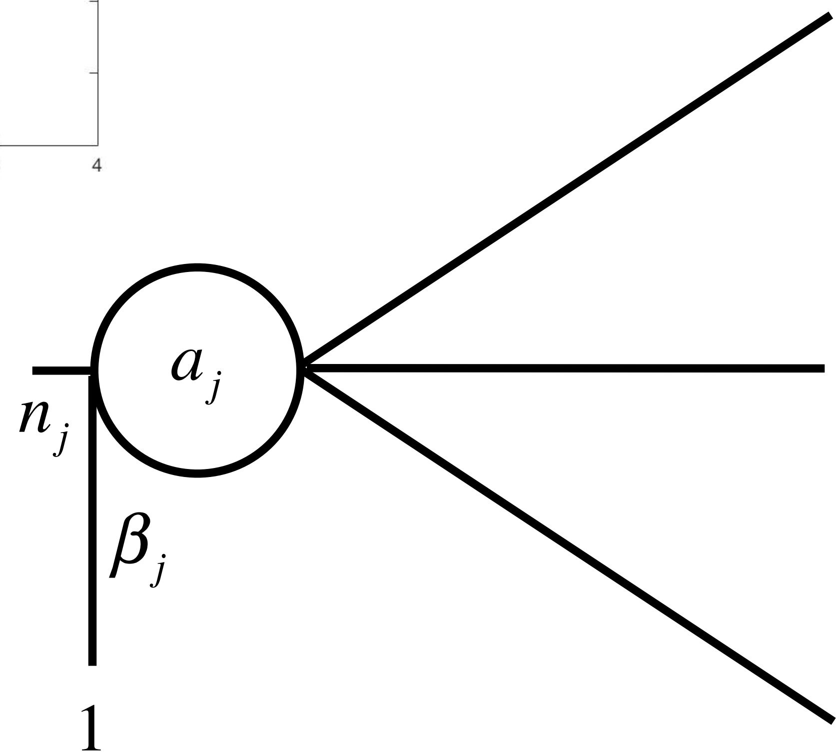
$$a_j = \frac{1}{1+\exp(-n_j)}$$

<u>from homework</u>

$$a_j = \frac{L}{1+\exp(-k(n_j-\theta_j))}$$



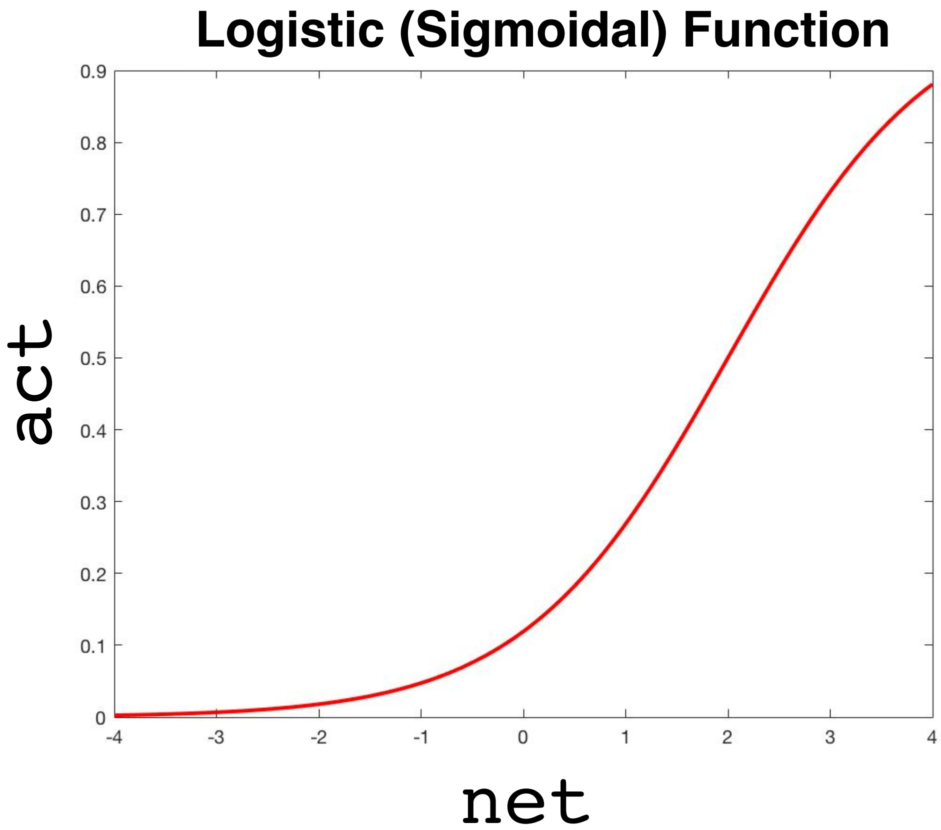
$$a_j = \frac{1}{1+\exp(-(n_j+\beta_j))}$$



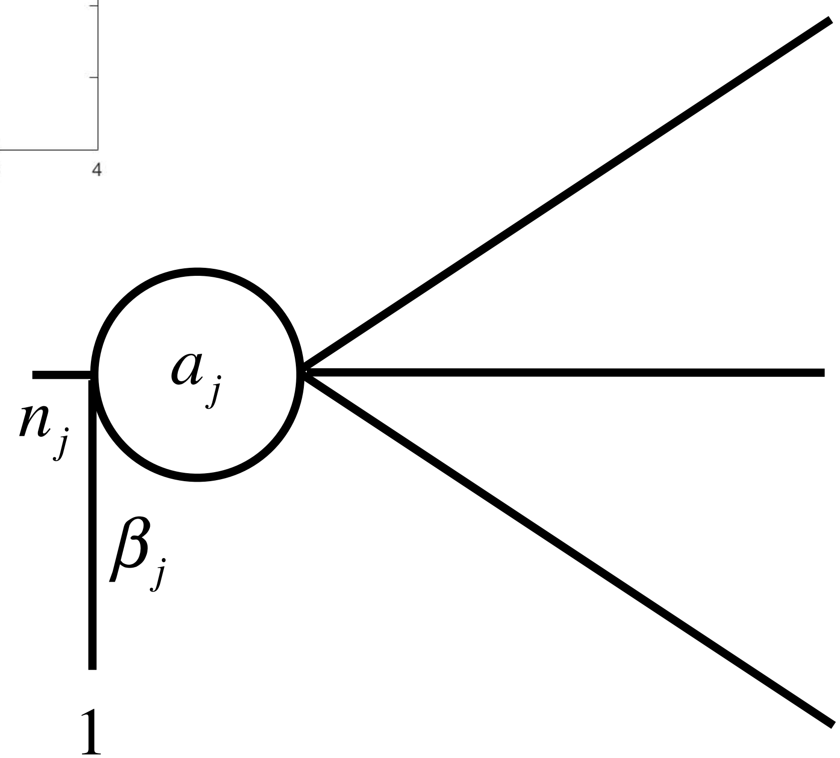
<u>from homework</u>

$$a_j = \frac{L}{1+\exp(-k(n_j-(-\beta_j)))}$$

Idealized Neuron



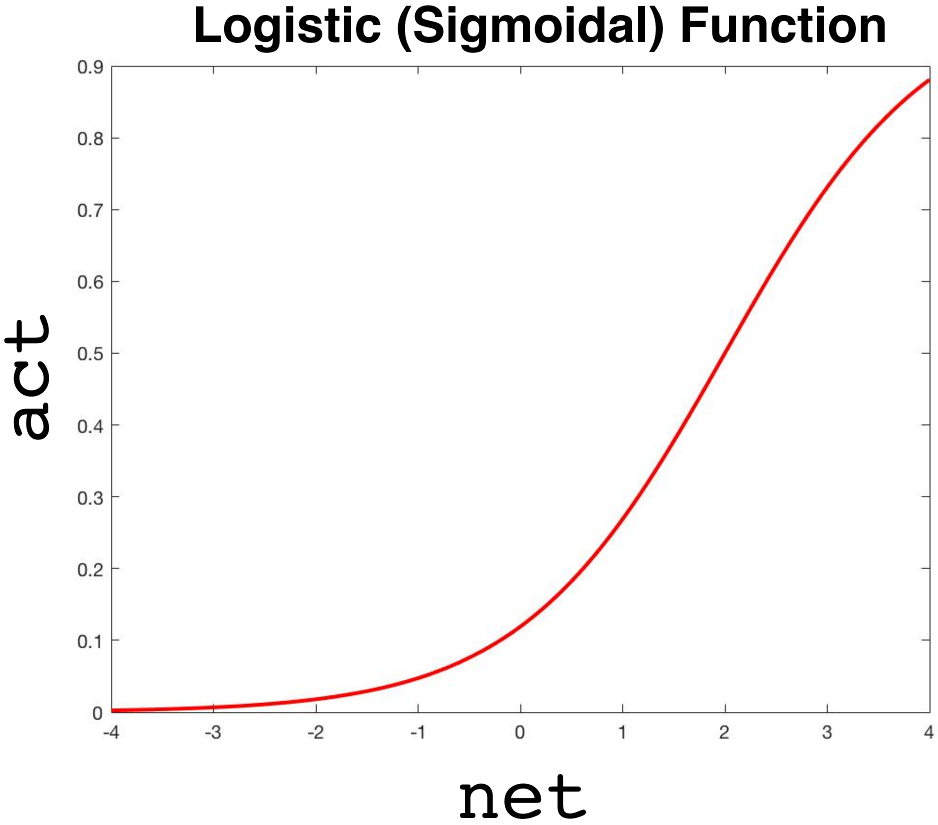
$$a_j = \frac{1}{1+\exp(-(n_j+\beta_j))}$$



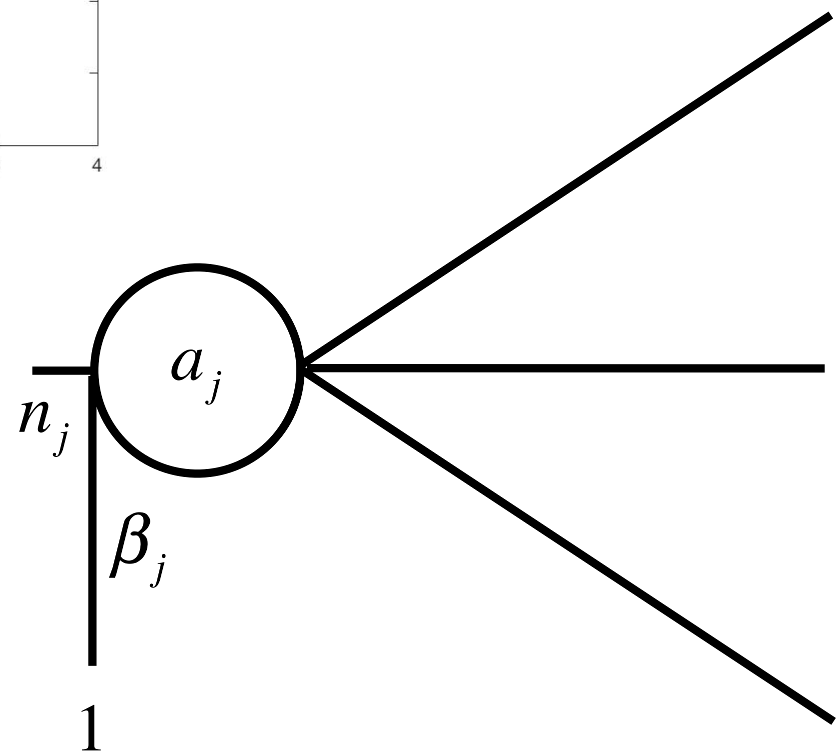
<u>from homework</u>

$$a_j = \frac{L}{1+\exp(-k(n_j-(-\beta_j)))}$$

k term is like multiplying all input weights and bias by k



$$a_j = \frac{1}{1+\exp(-(n_j+\beta_j))}$$

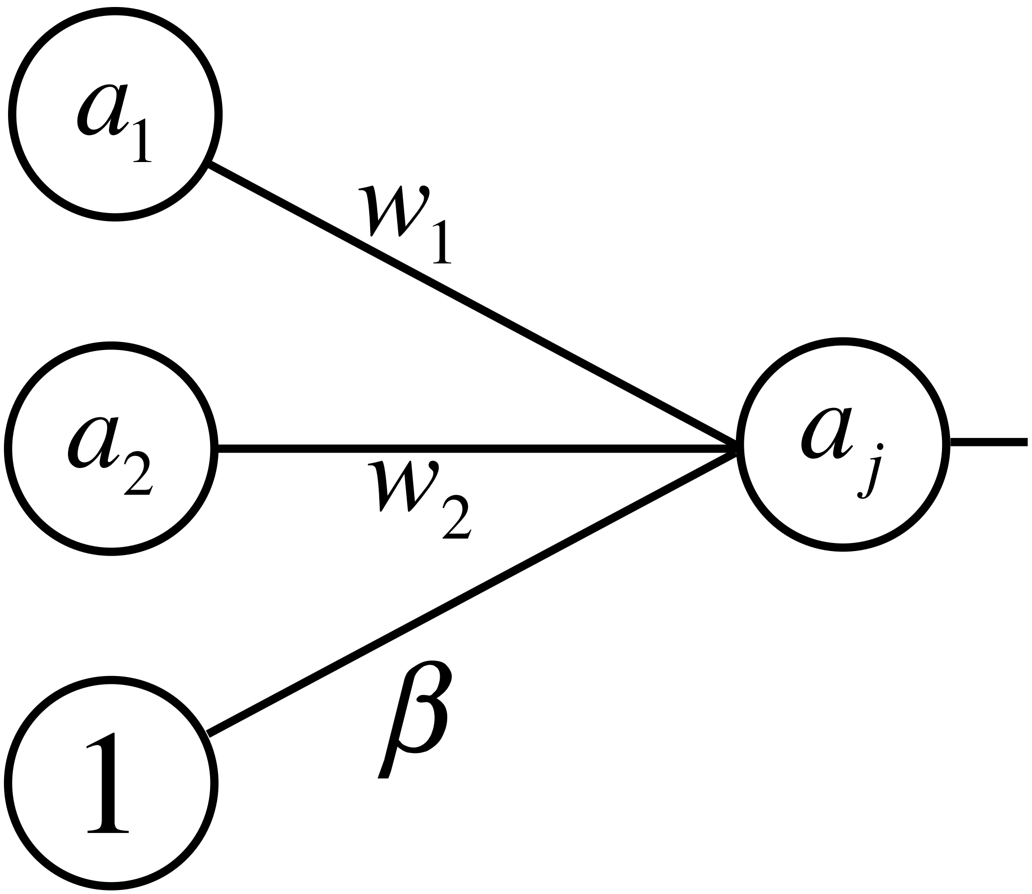


<u>from homework</u>

$$a_j = \frac{L}{1+\exp(-k(n_j-(-\beta_j)))}$$

L term is like multiplying all output weights by L

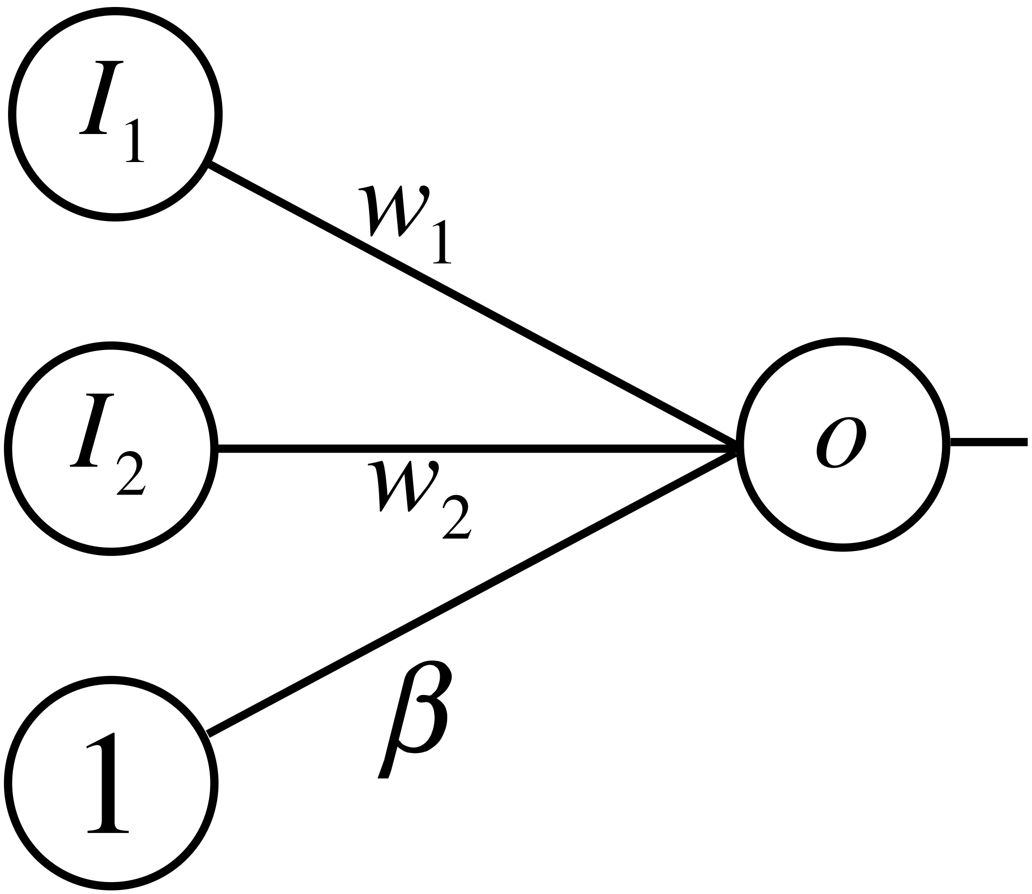
Example of a Simple Neural Network



Example of a Simple Neural Network

inputs can be
sensory, perceptual,
or abstract features

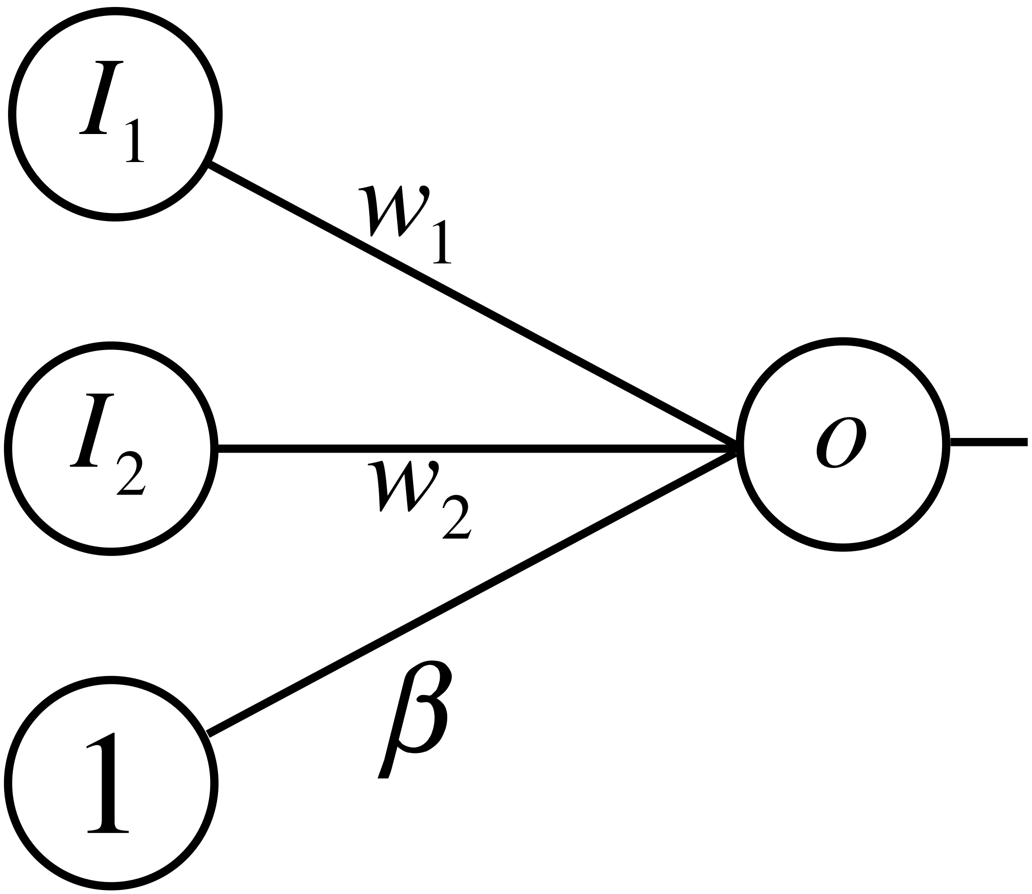
they can be
discrete or
continuous values



I_1	I_2	o
0	0	0
1	0	0
0	1	0
1	1	1

what is this
computation?

Example of a Simple Neural Network



I_1	I_2	o
0	0	0
1	0	0
0	1	0
1	1	1

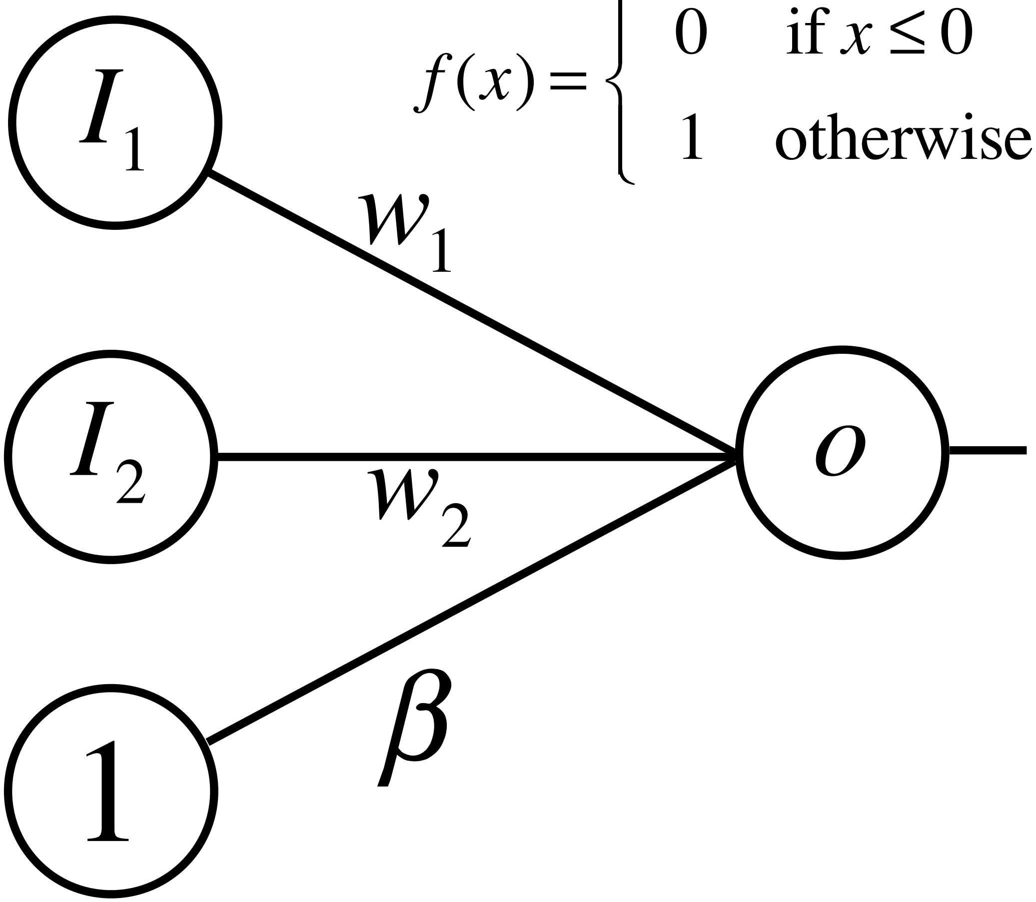
logical AND

Example of a Simple Neural Network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{otherwise} \end{cases}$$



I_1	I_2	o
0	0	0
1	0	0
0	1	0
1	1	1

logical AND

Example of a Simple Neural Network

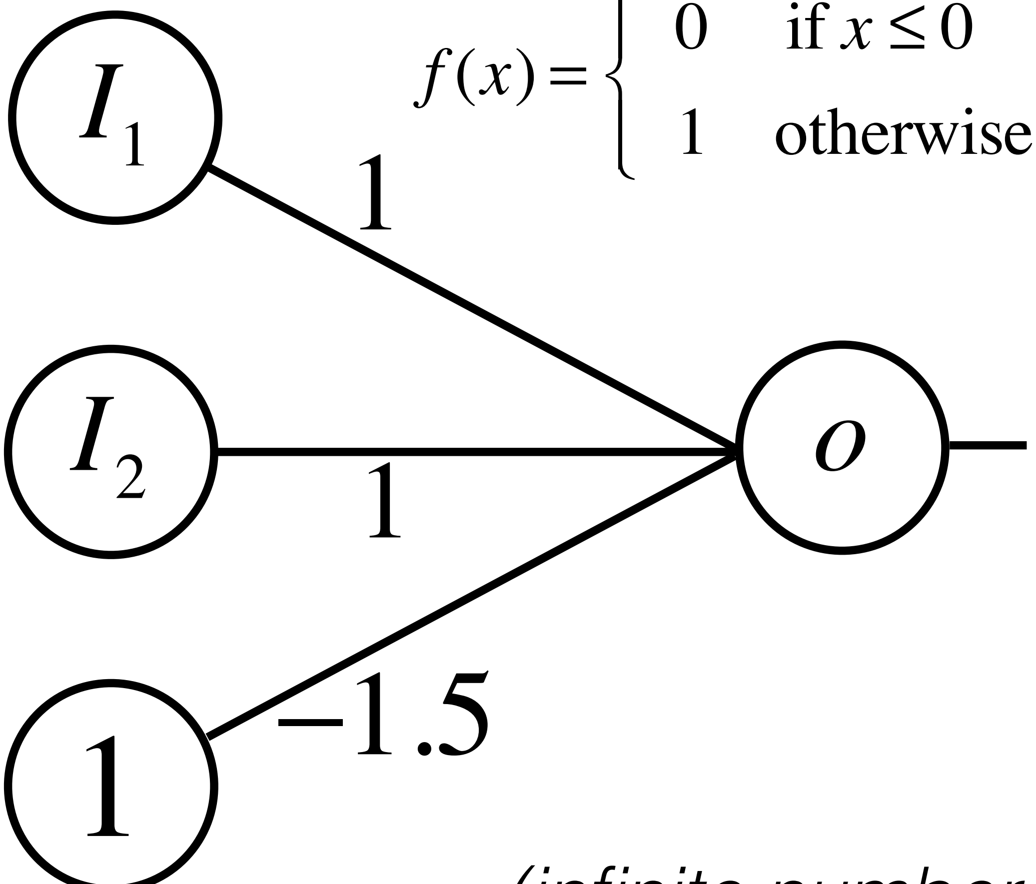
$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

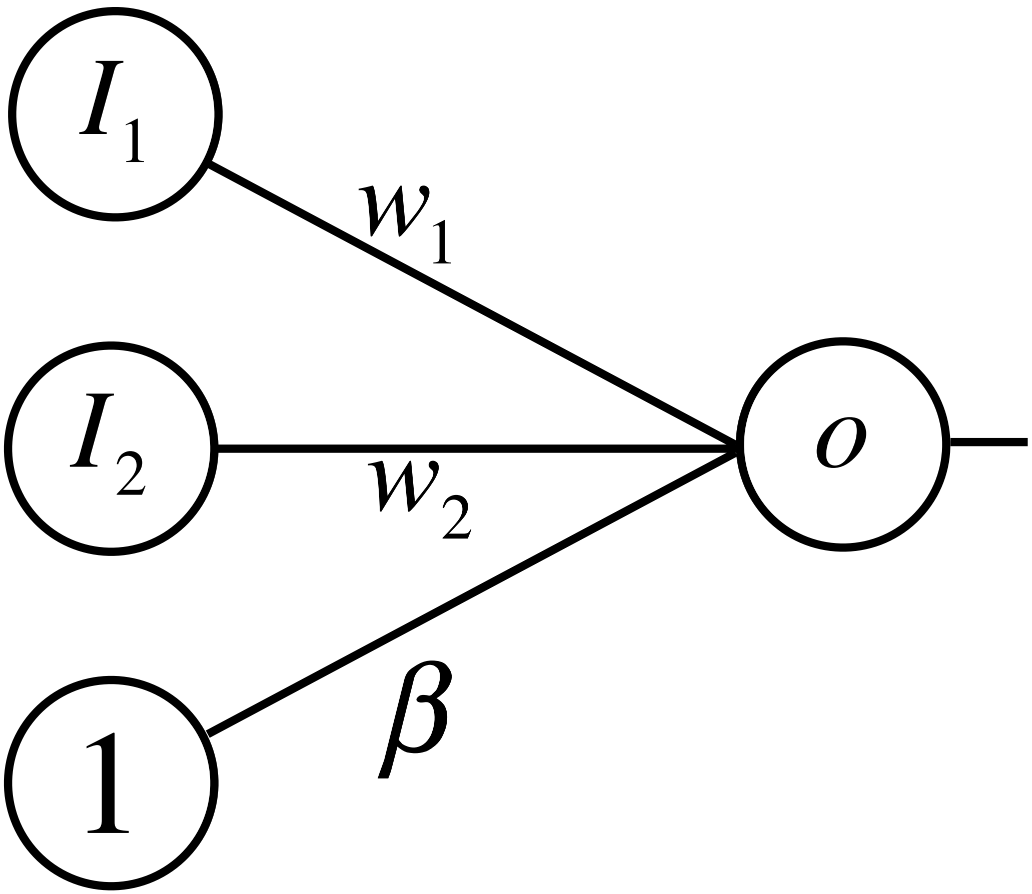
I_1	I_2	o
0	0	0
1	0	0
0	1	0
1	1	1

logical AND



(infinite number of solutions)

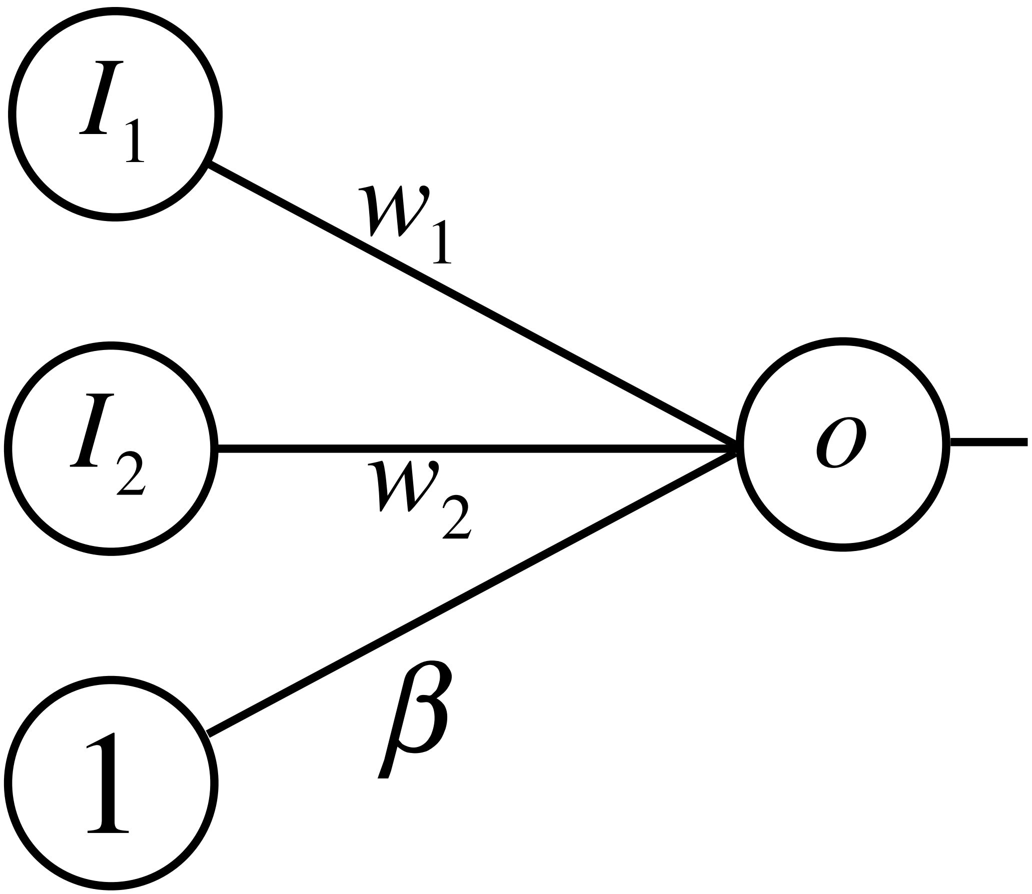
Example of a Simple Neural Network



I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	1

what is this computation?

Example of a Simple Neural Network



I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	1

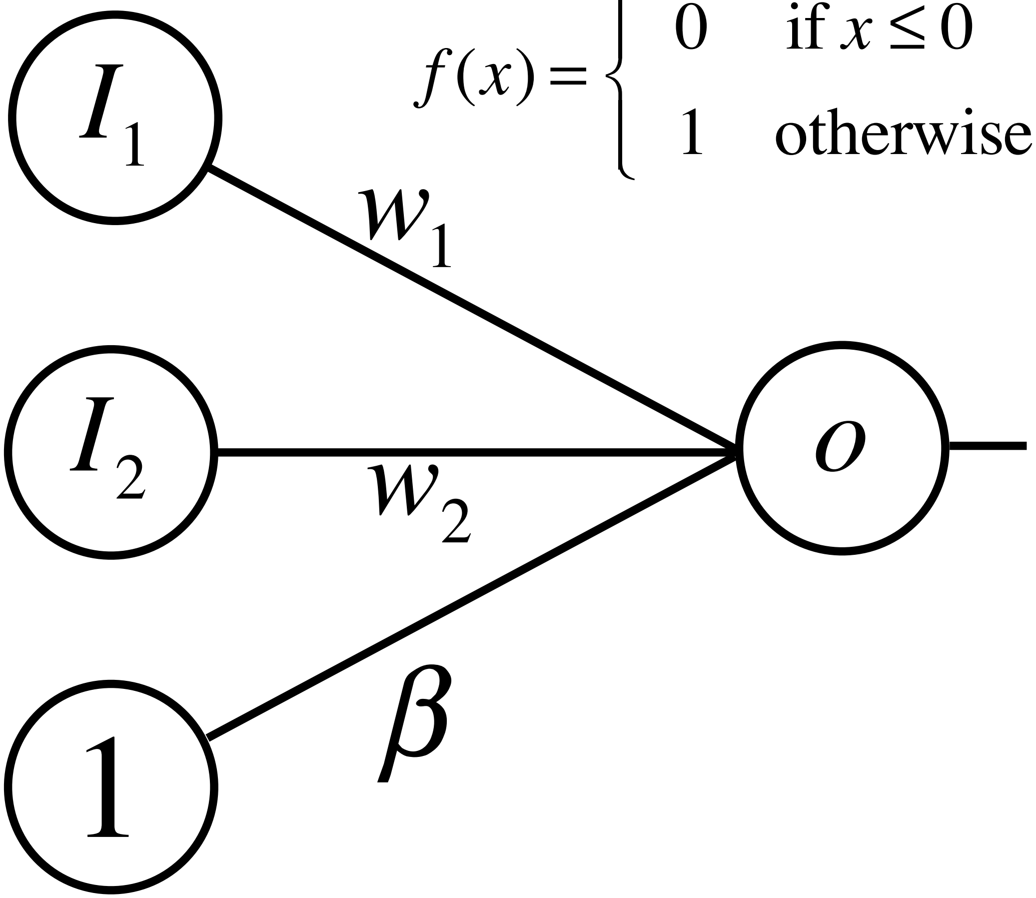
logical
OR

Example of a Simple Neural Network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases}$$



I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	1

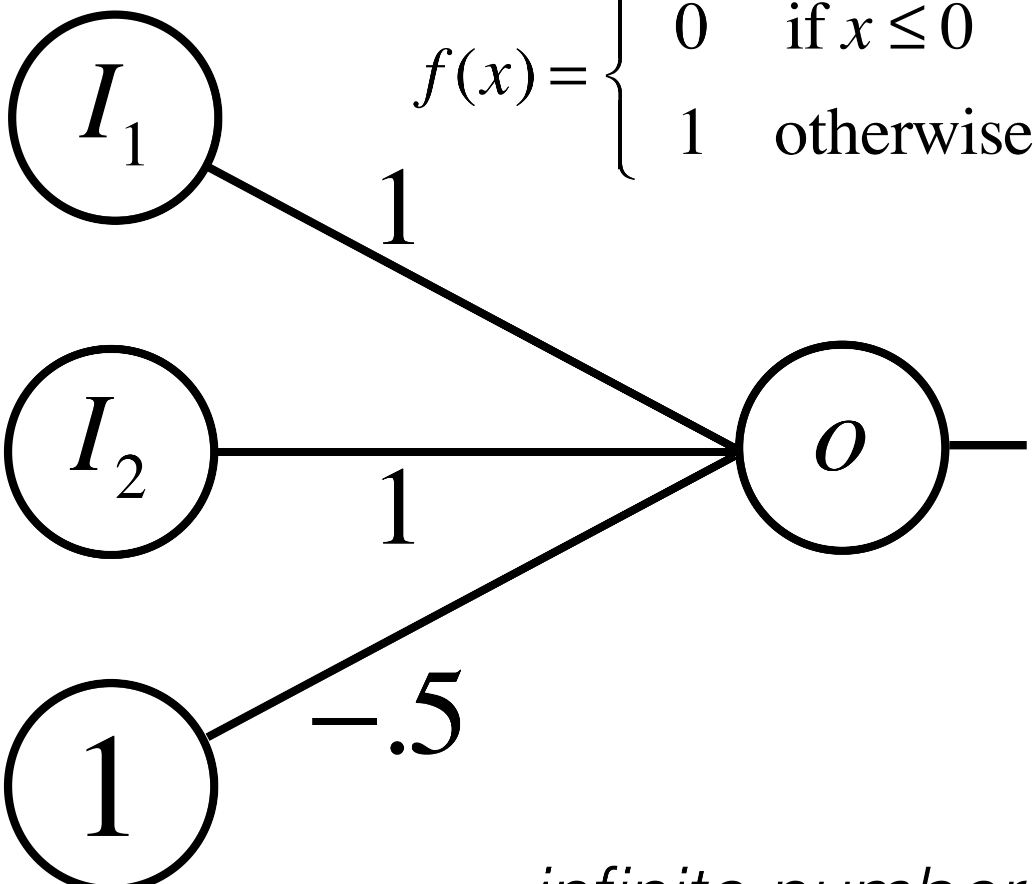
logical
OR

Example of a Simple Neural Network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

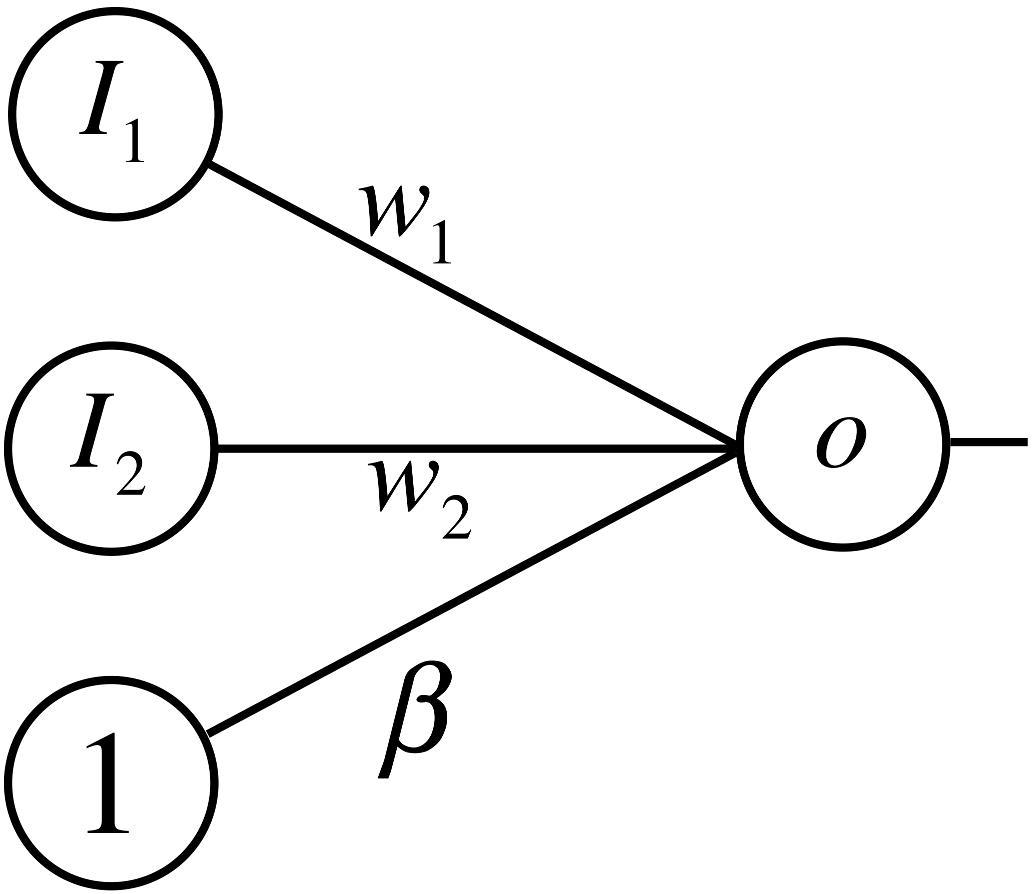


infinite number of solutions

I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	1

logical
OR

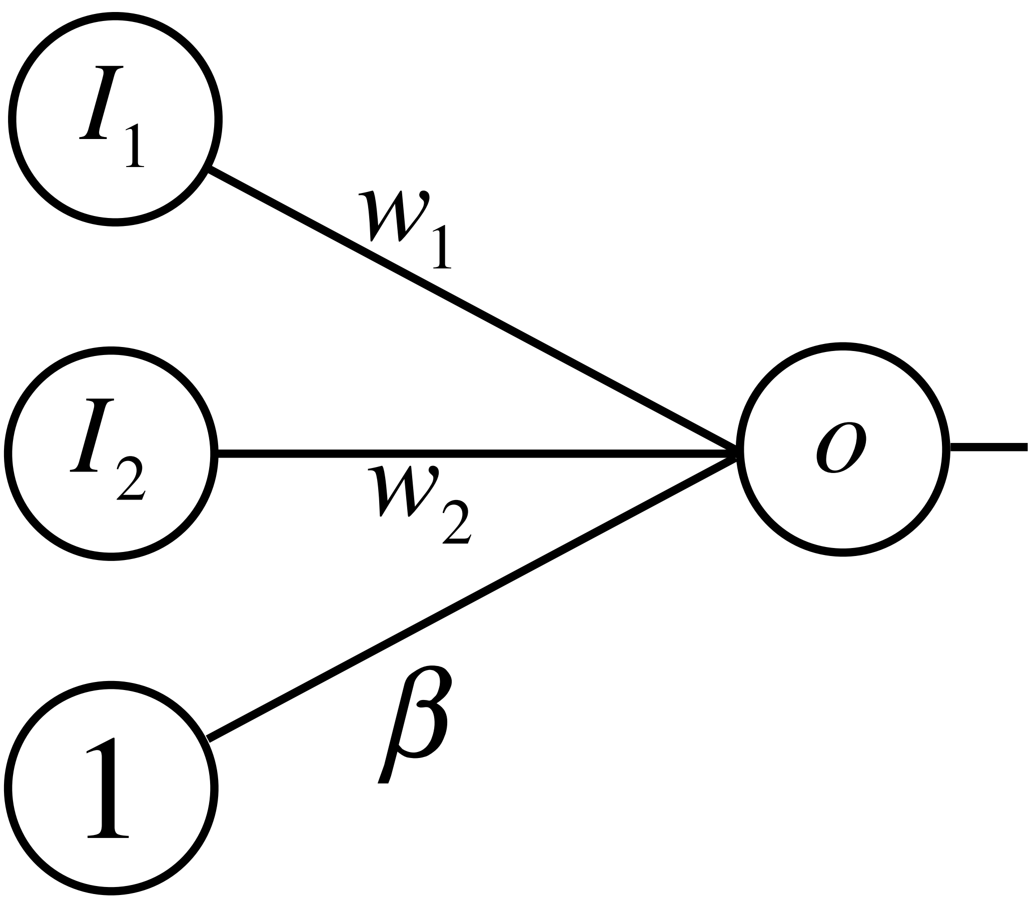
Example of a Simple Neural Network



I_1	I_2	o
0	0	1
1	0	0
0	1	0
1	1	1

what is this computation?

Example of a Simple Neural Network



I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	0

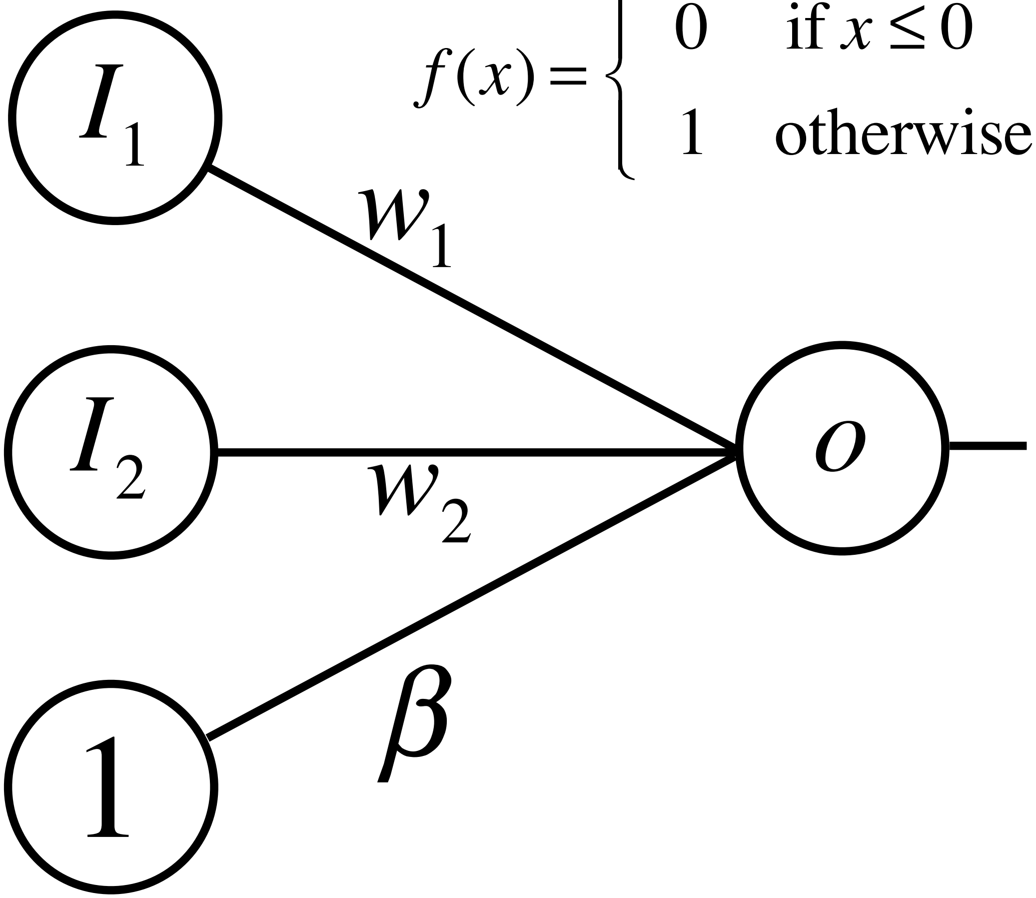
logical
XOR

Example of a Simple Neural Network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases}$$



I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	0

logical XOR

Example of a Simple Neural Network

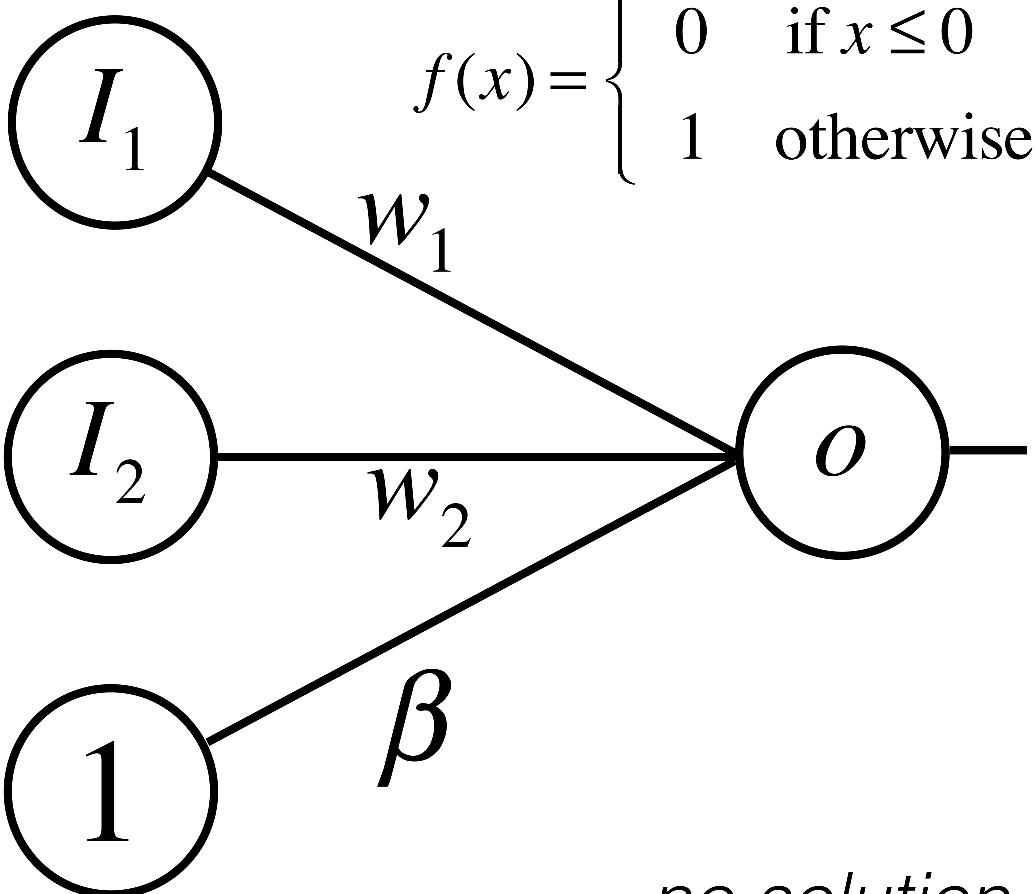
$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	0

logical XOR



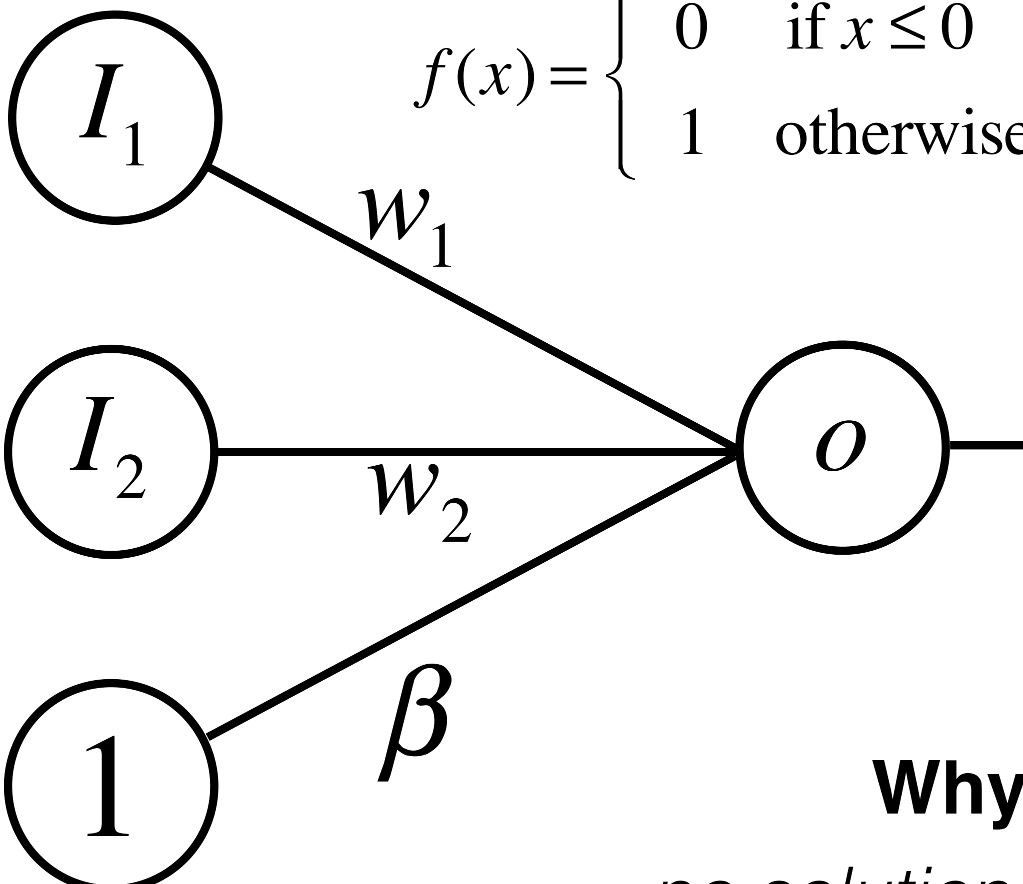
no solution exists!!!

Example of a Simple Neural Network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases}$$



I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	0

logical XOR

Why?

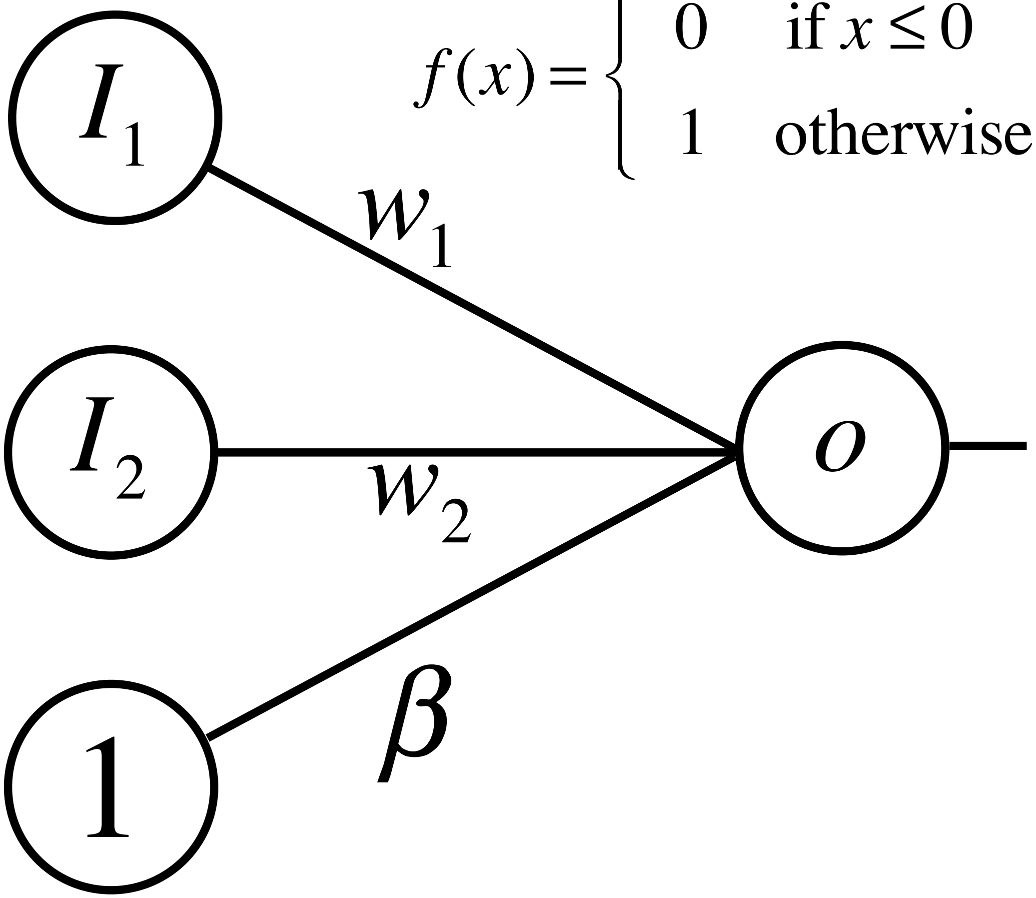
no solution exists!!!

Example of a Simple Neural Network

$$o = f\left(\sum_i I_i w_i + \beta\right)$$

what values of w_1, w_2, and β

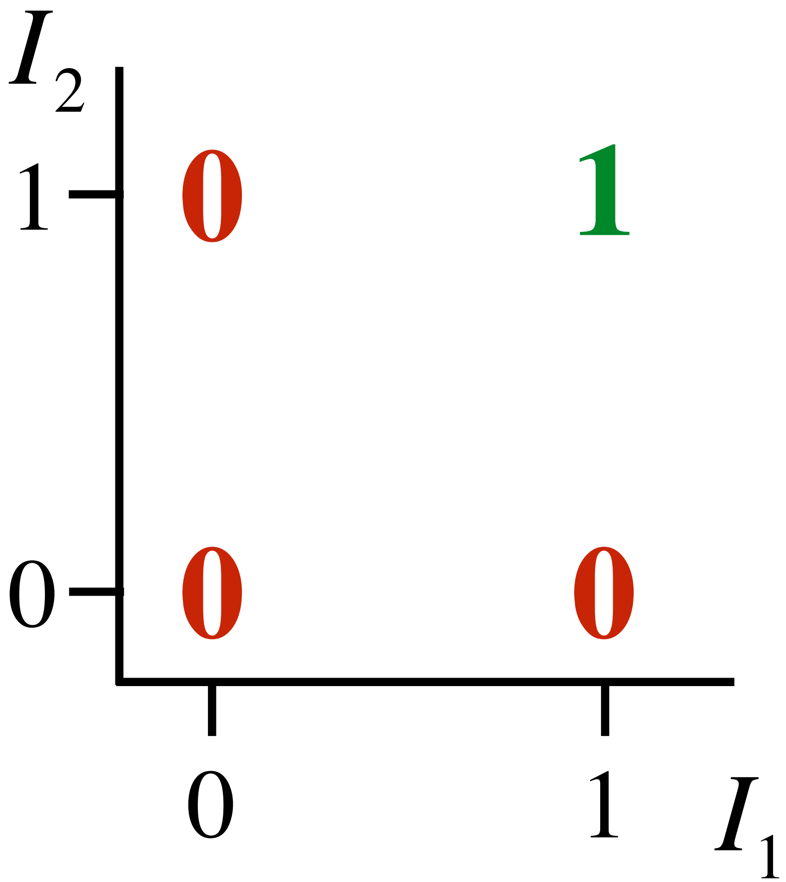
$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases}$$



I_1	I_2	o
0	0	0
1	0	0
0	1	0
1	1	1

logical AND

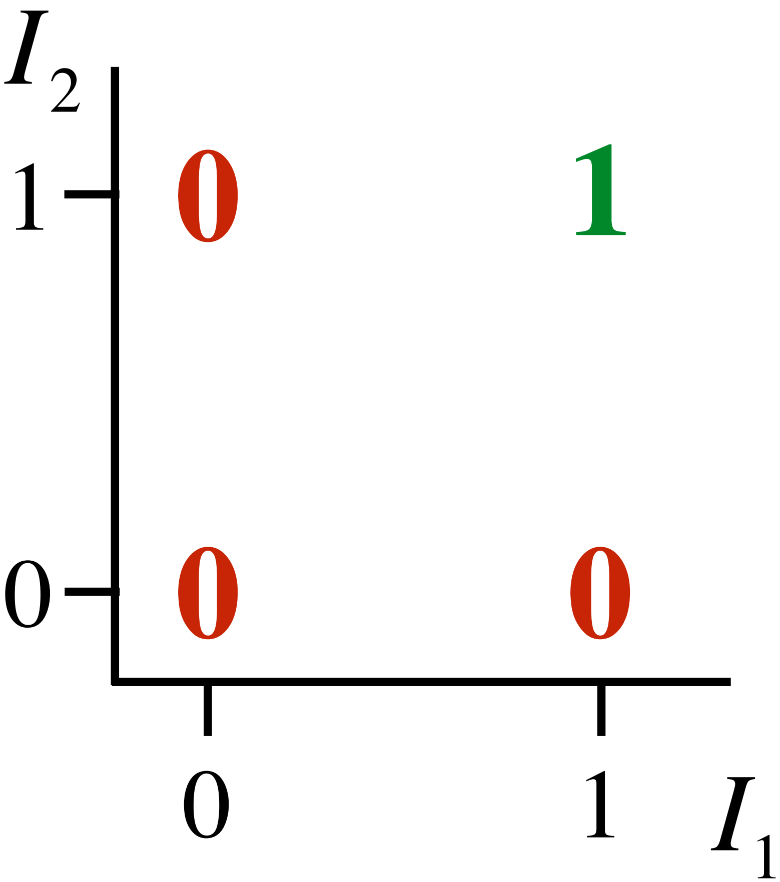
Example of a Simple Neural Network



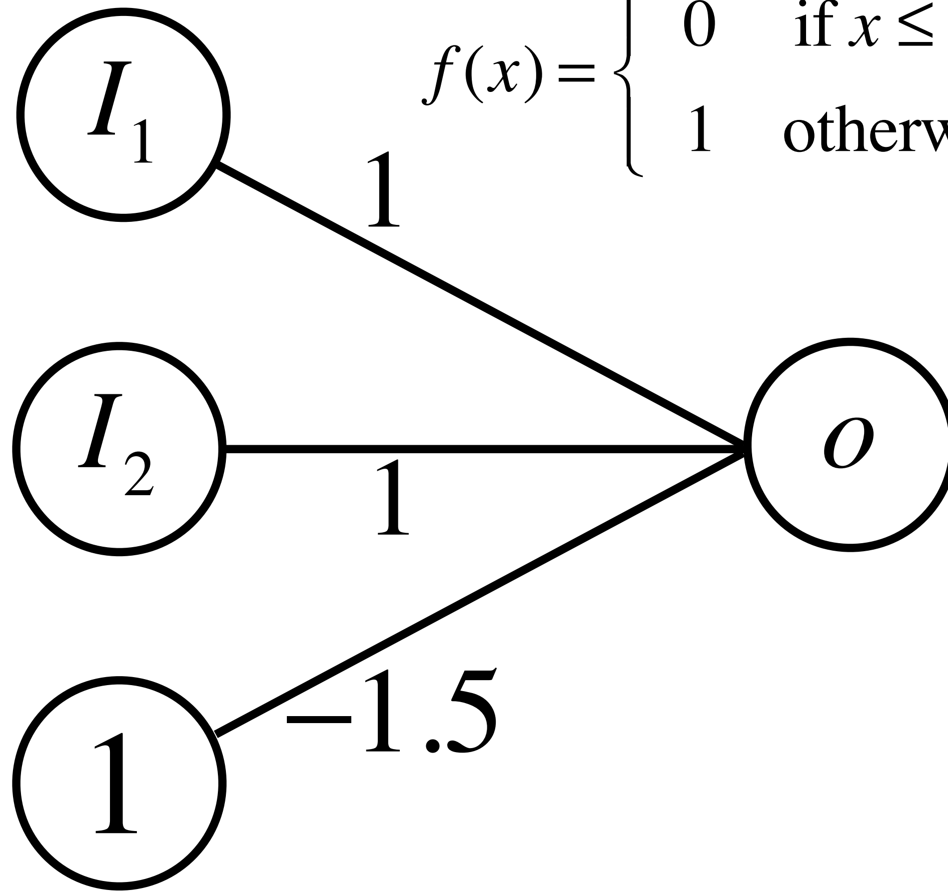
I_1	I_2	o
0	0	0
1	0	0
0	1	0
1	1	1

logical AND

Example of a Simple Neural Network

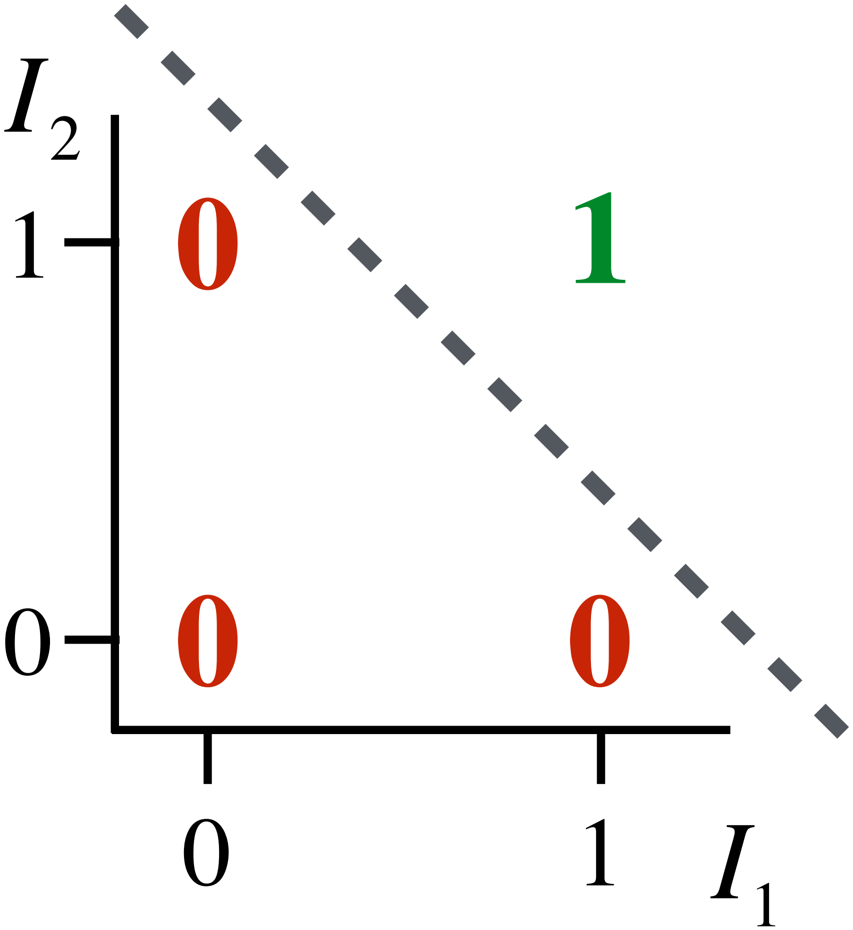


$$f(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{otherwise} \end{cases}$$

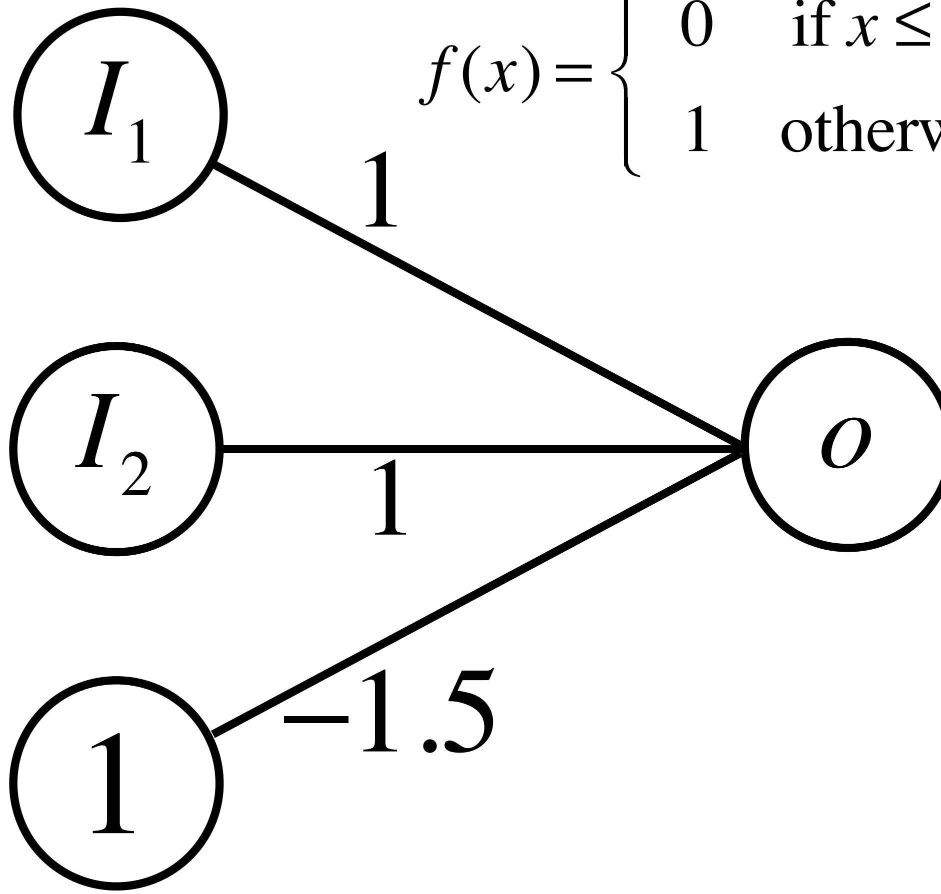


$$1 \times I_1 + 1 \times I_2 + (-1.5) \times 1 = 0$$

Example of a Simple Neural Network



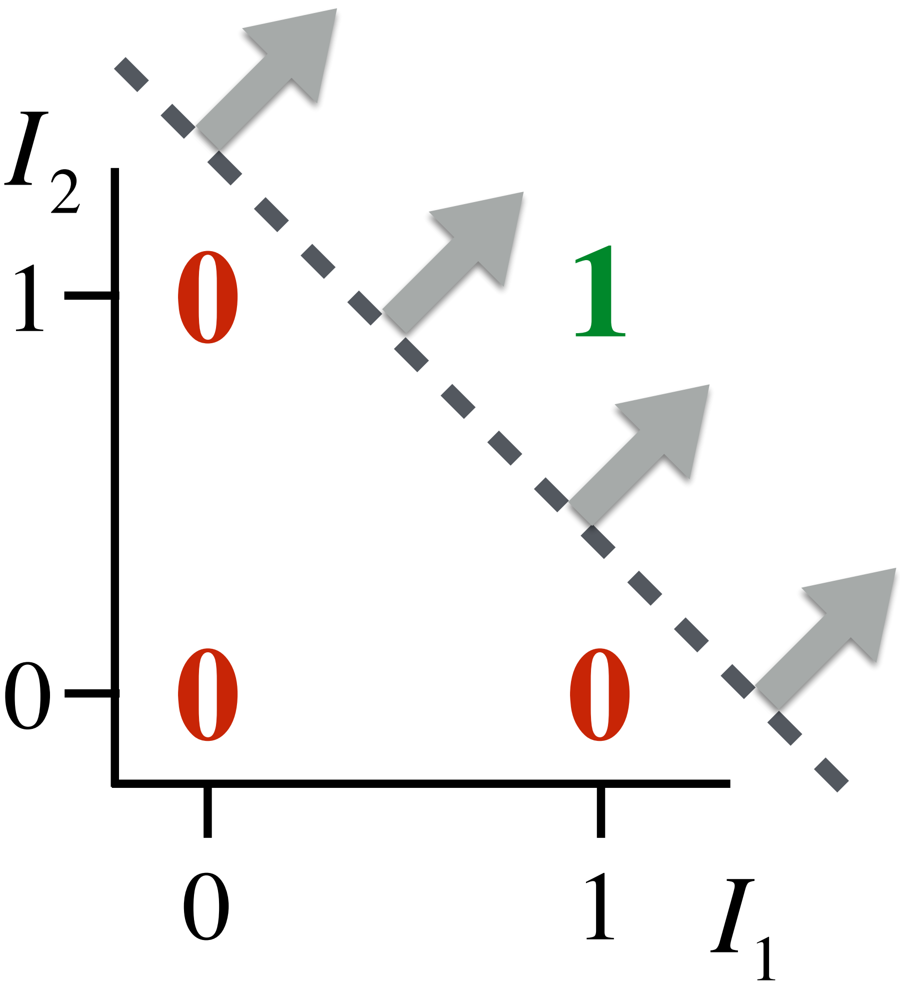
$$f(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{otherwise} \end{cases}$$

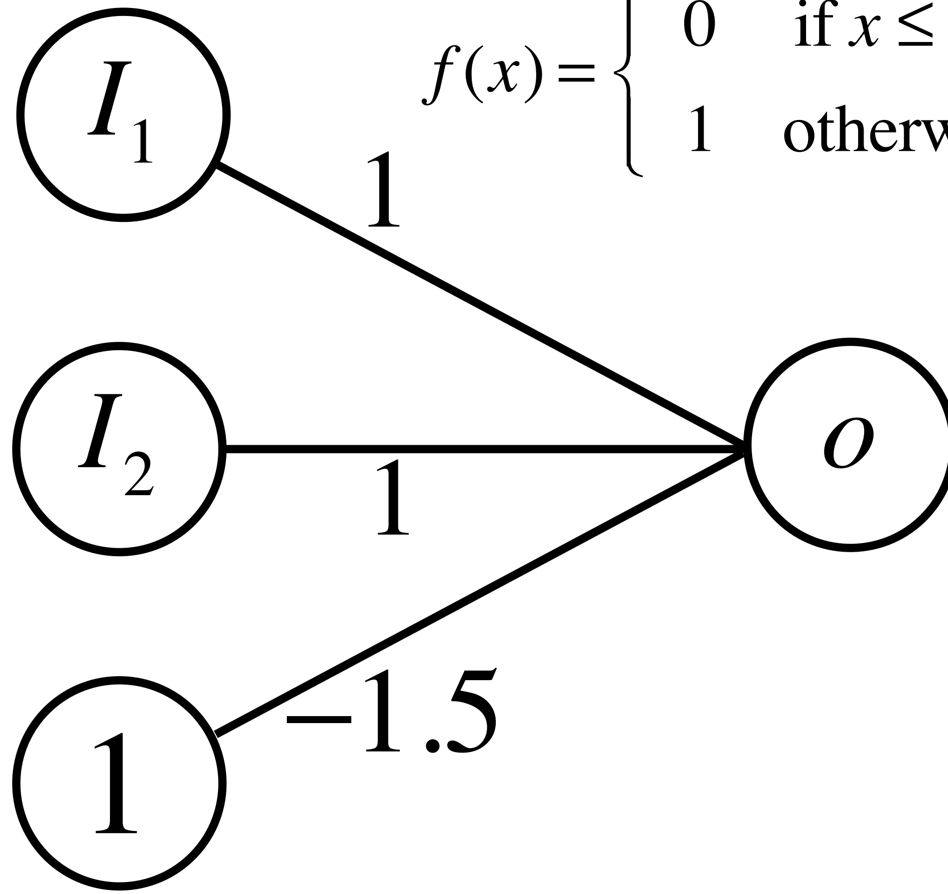


$$I_1 + I_2 - 1.5 = 0$$

$$I_2 = -I_1 + 1.5$$

Example of a Simple Neural Network



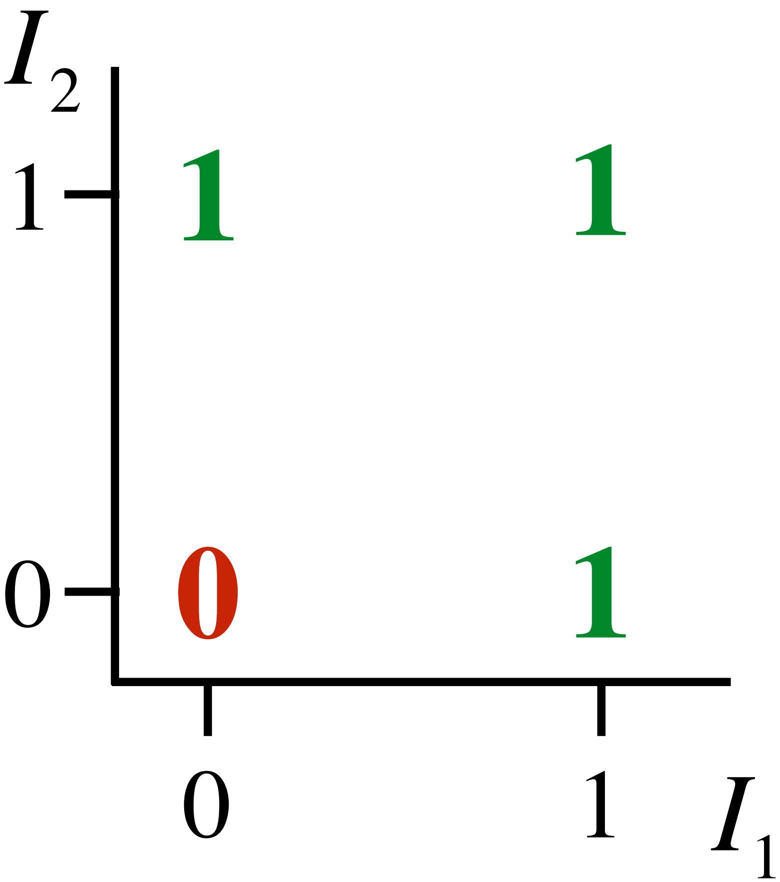


$$f(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{otherwise} \end{cases}$$

$$I_1 + I_2 - 1.5 = 0$$

$$I_2 = -I_1 + 1.5$$

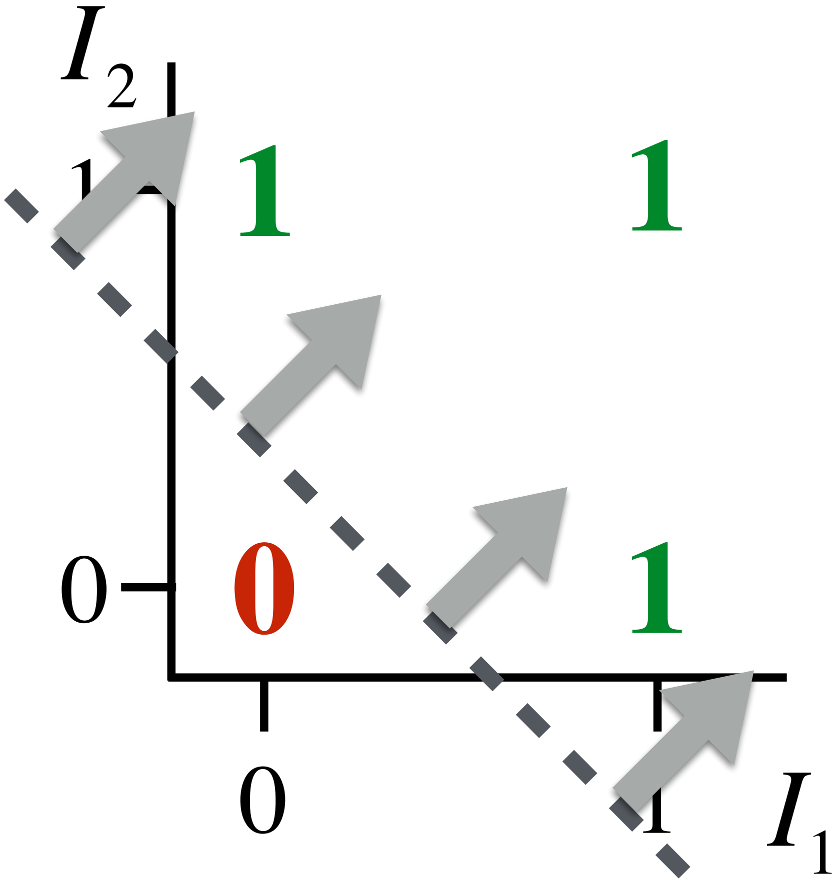
Example of a Simple Neural Network



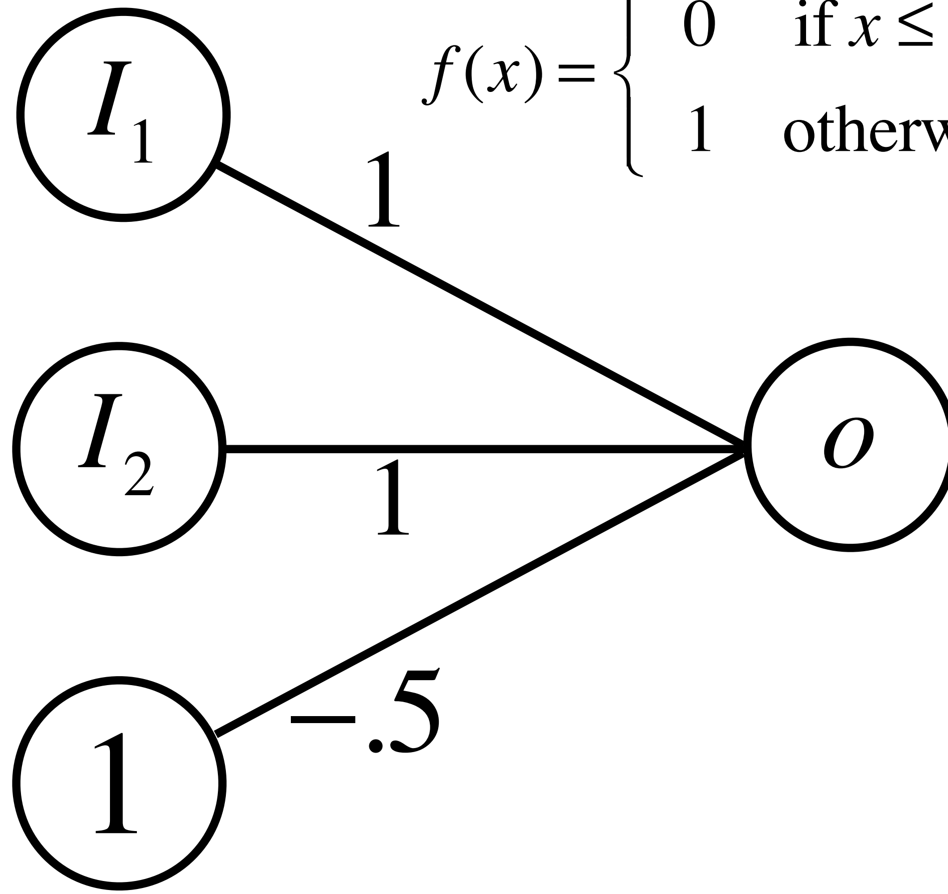
I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	1

logical OR

Example of a Simple Neural Network



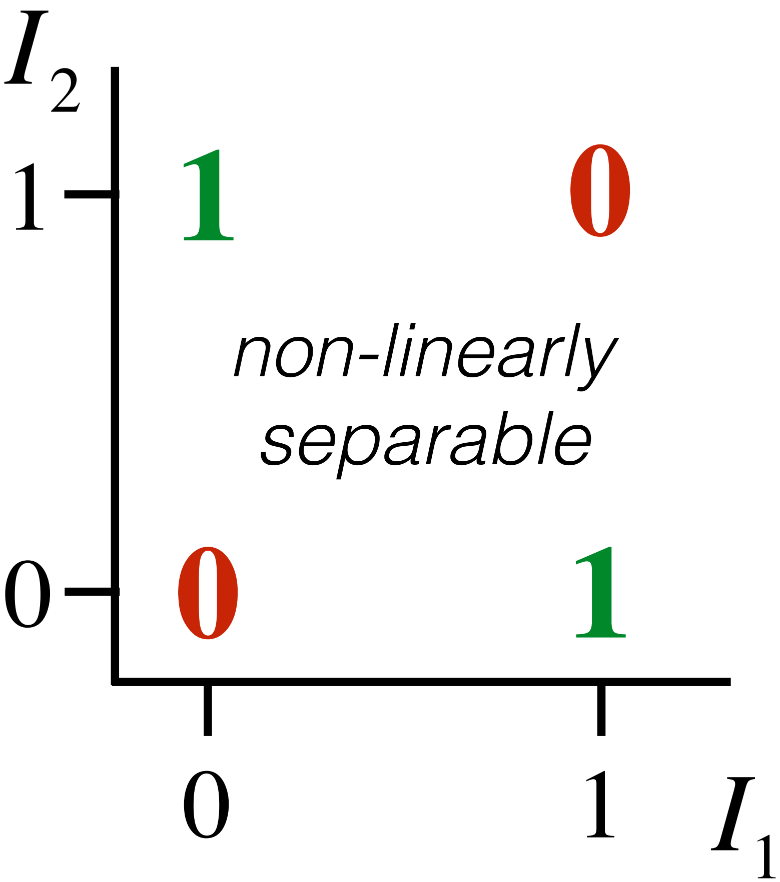
$$f(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{otherwise} \end{cases}$$



$$1 \times I_1 + 1 \times I_2 - .5 = 0$$

$$I_2 = -I_1 + .5$$

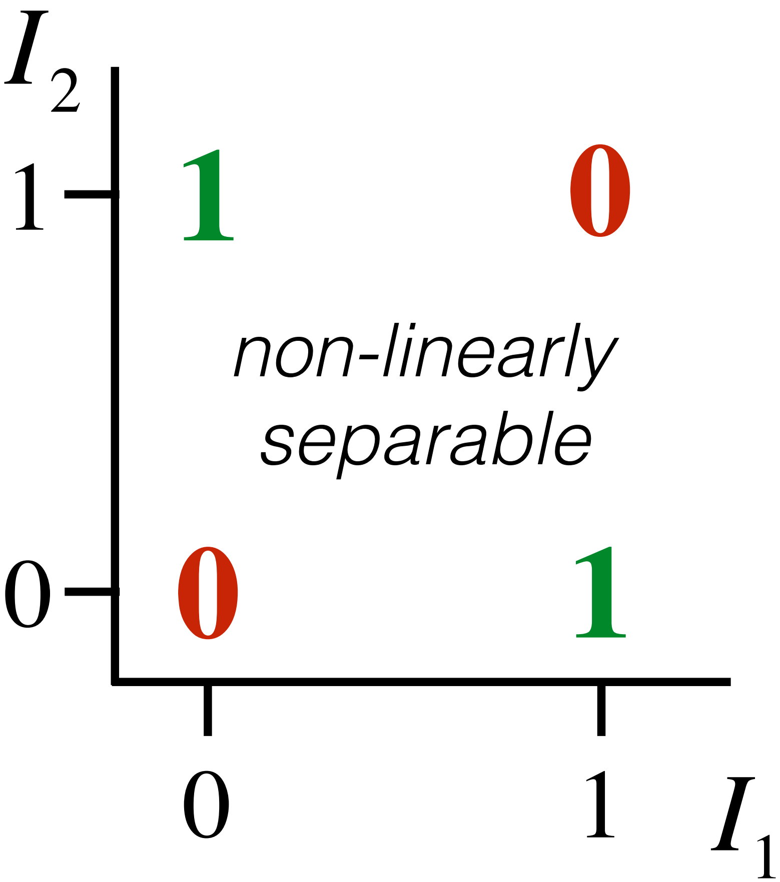
Example of a Simple Neural Network



I_1	I_2	o
0	0	0
1	0	1
0	1	1
1	1	0

logical
XOR

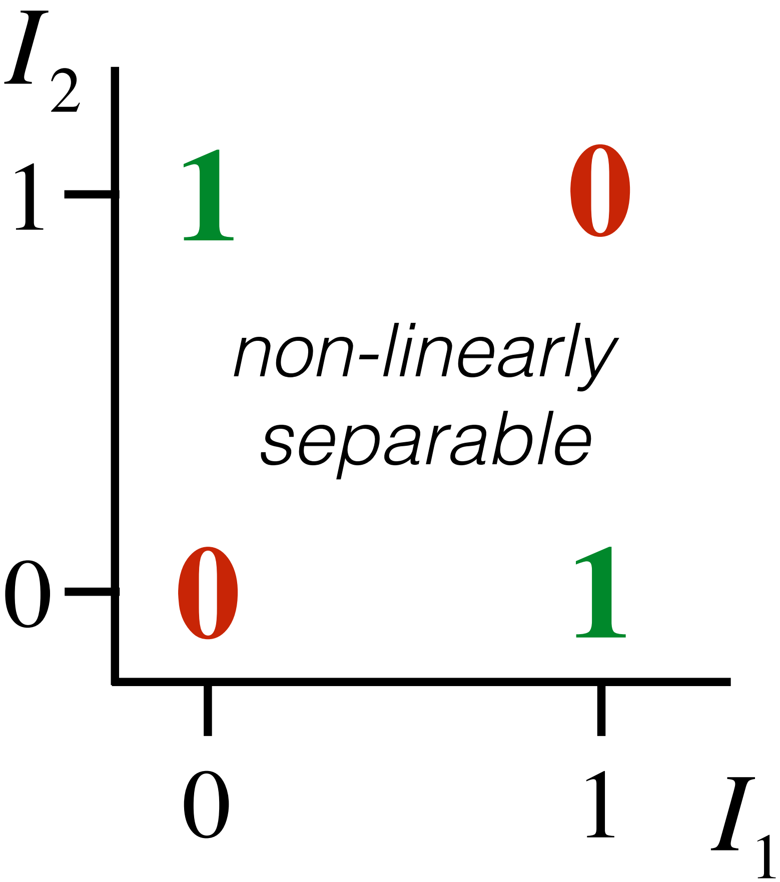
Example of a Simple Neural Network



I_1	I_2	o
$w_1 \times 0 + w_2 \times 0 + \beta < 0$		
1	0	1
0	1	1
1	1	0

logical
XOR

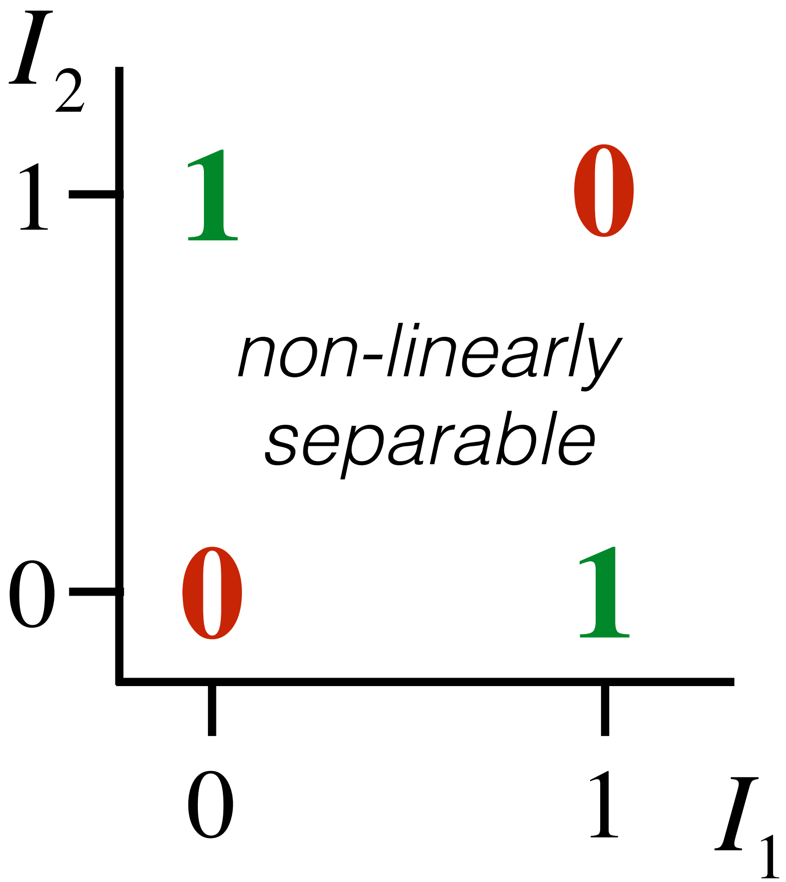
Example of a Simple Neural Network



I_1	I_2	o
$w_1 \times 0 + w_2 \times 0 + \beta < 0$		
$w_1 \times 1 + w_2 \times 0 + \beta \geq 0$		
0	1	0
1	1	1

logical XOR

Example of a Simple Neural Network



I_1	I_2	o

$$w_1 \times 0 + w_2 \times 0 + \beta < 0$$

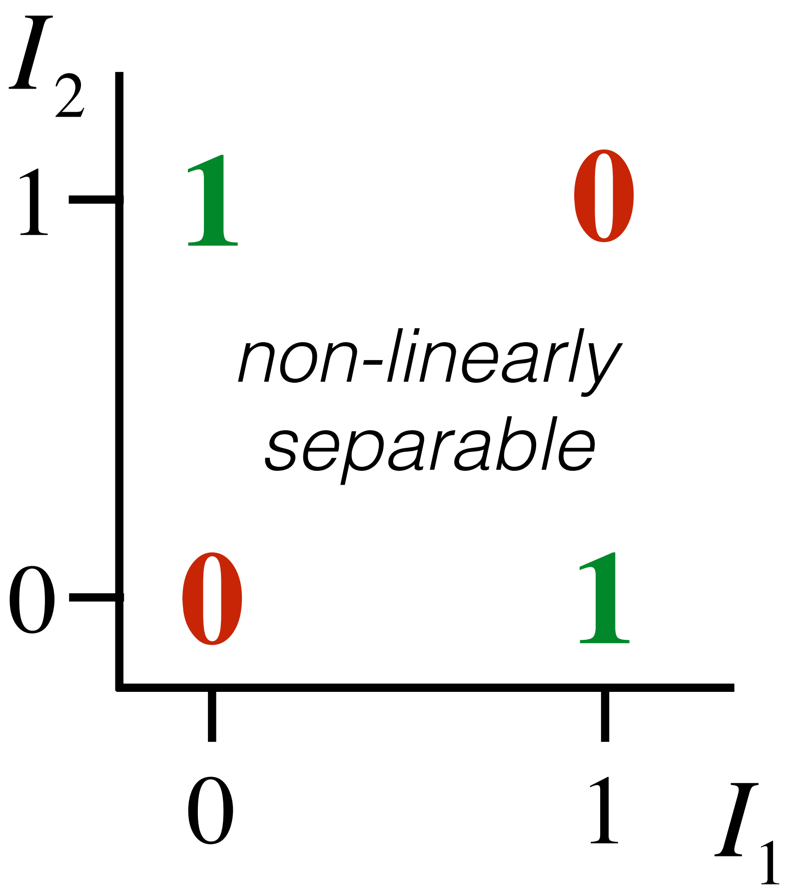
$$w_1 \times 1 + w_2 \times 0 + \beta \geq 0$$

$$w_1 \times 0 + w_2 \times 1 + \beta \geq 0$$

1	1	1

logical XOR

Example of a Simple Neural Network

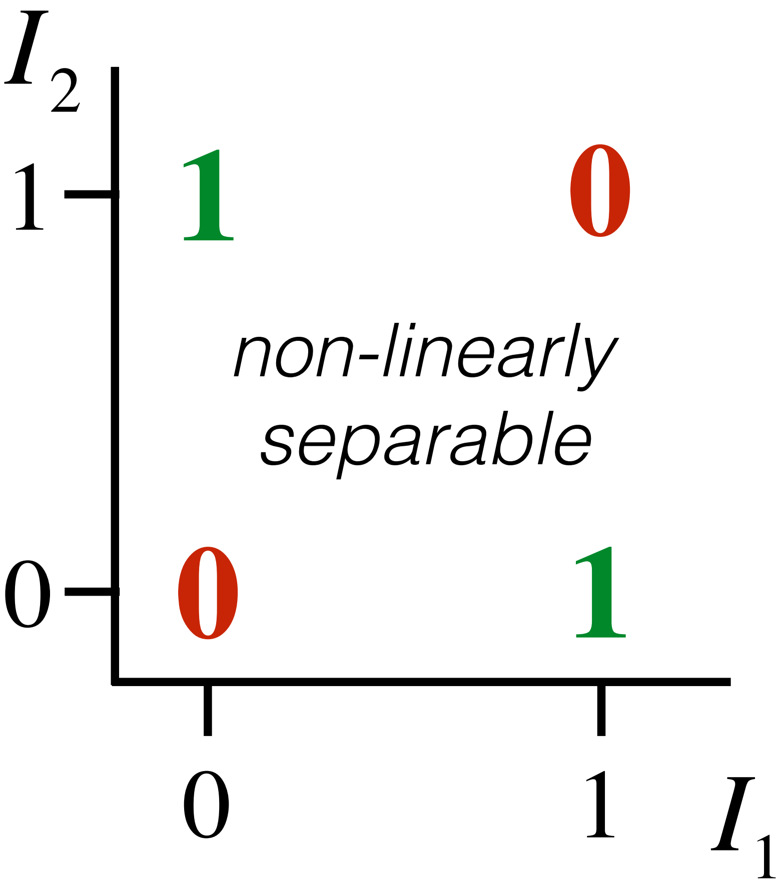


I_1	I_2	o

$$w_1 \times 0 + w_2 \times 0 + \beta < 0$$
$$w_1 \times 1 + w_2 \times 0 + \beta \geq 0$$
$$w_1 \times 0 + w_2 \times 1 + \beta \geq 0$$
$$w_1 \times 1 + w_2 \times 1 + \beta < 0$$

logical
XOR

Example of a Simple Neural Network



I_1	I_2	o

$$\beta < 0$$

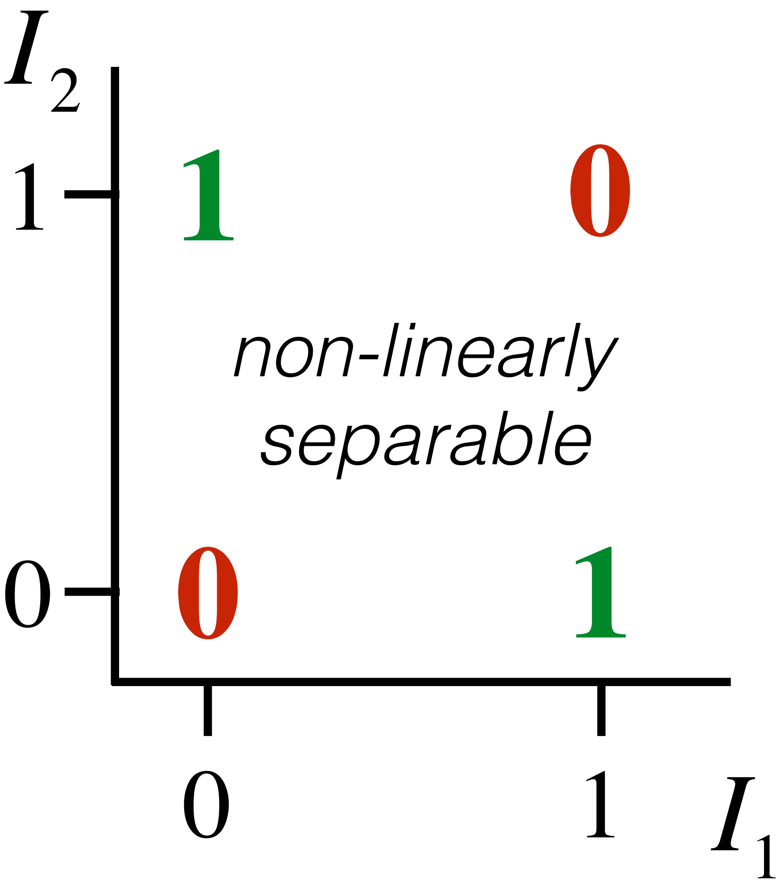
$$w_1 + \beta \geq 0$$

$$w_2 + \beta \geq 0$$

$$w_1 + w_2 + \beta < 0$$

mutually
contradictory
(*convince yourself*)

Example of a Simple Neural Network



I_1	I_2	o

$$\beta < 0$$
$$w_1 + \beta \geq 0$$
$$w_2 + \beta \geq 0$$
$$w_1 + w_2 + \beta < 0$$

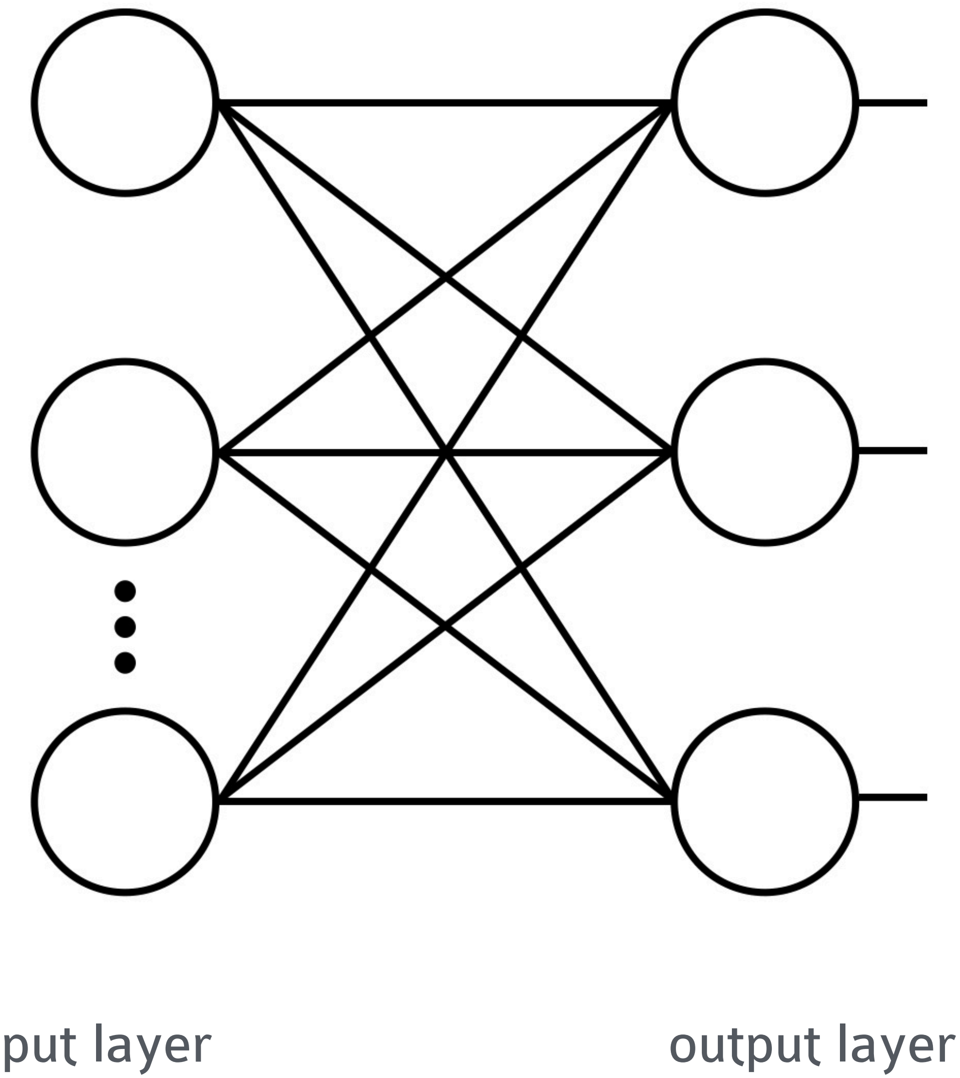
networks with multiple layers can solve this!!!
(the week after next)

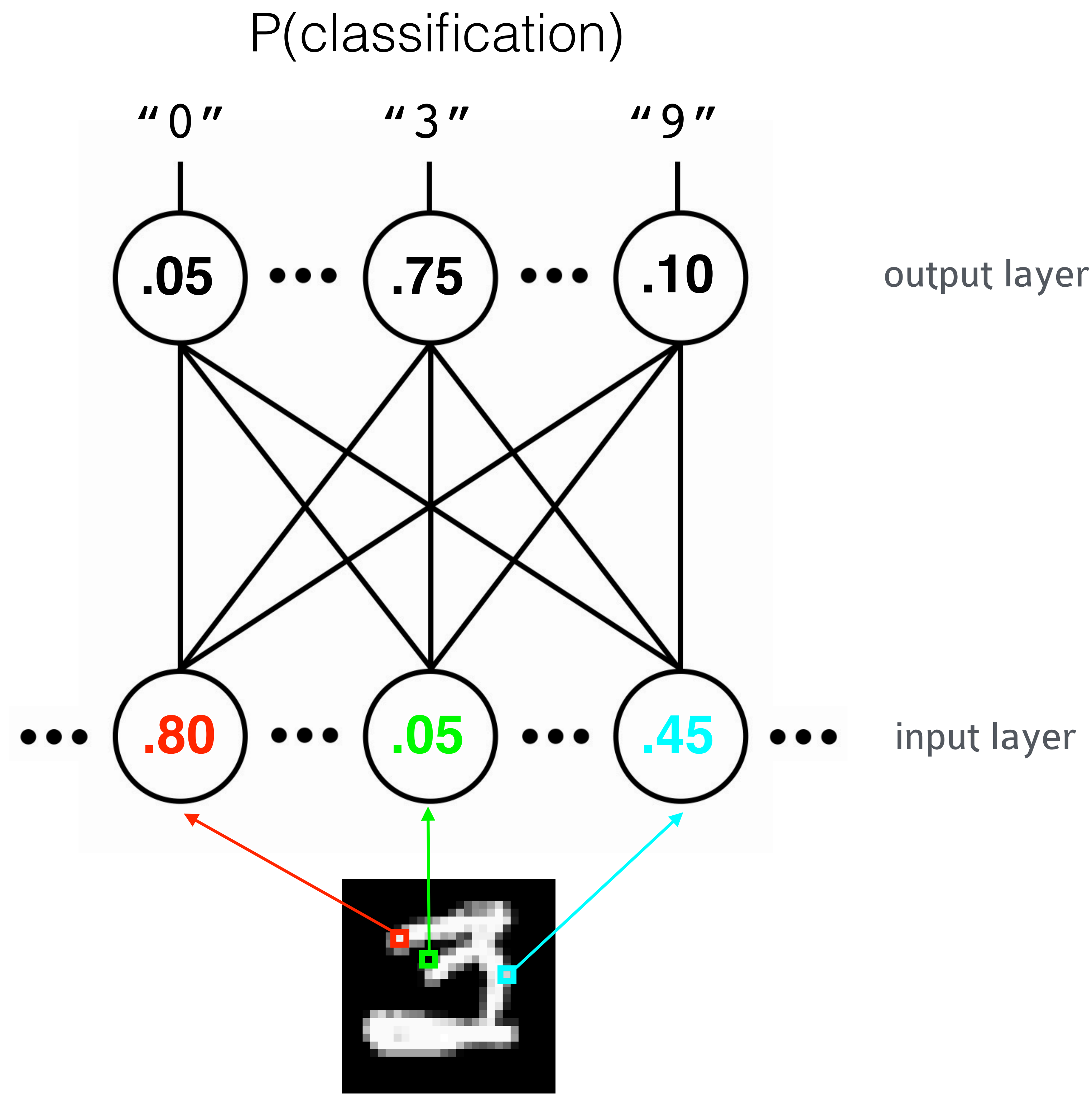
Example of a Simple Neural Network

&

Background for
Homework 3

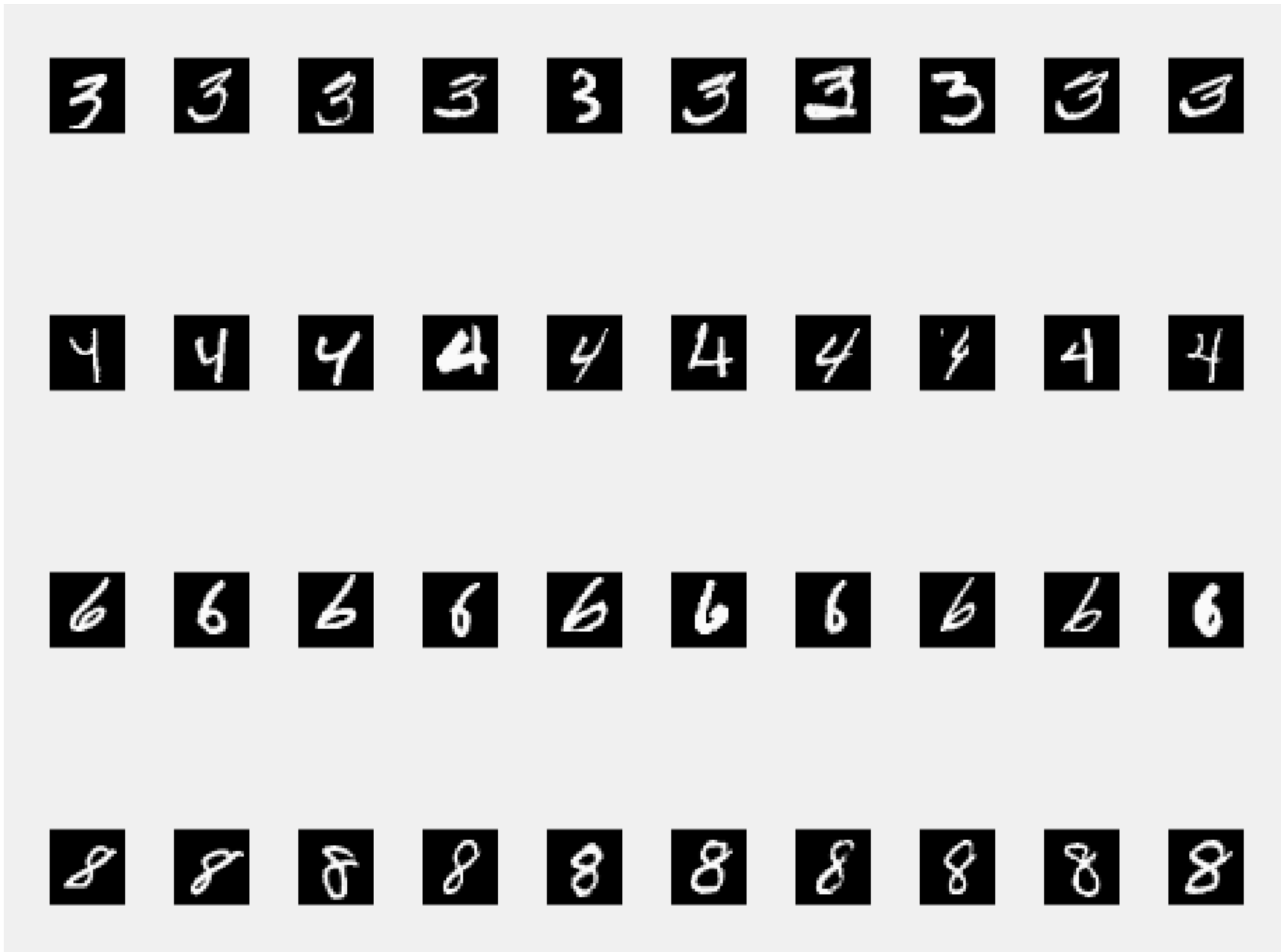
Example : Visual Digit Classification





P(classification)
"0"
"3"
"9"
.05
.75
.10
output layer
.80
.05
.45
input layer

- more practice using Python
- use a one-layer neural network (input and output layer)



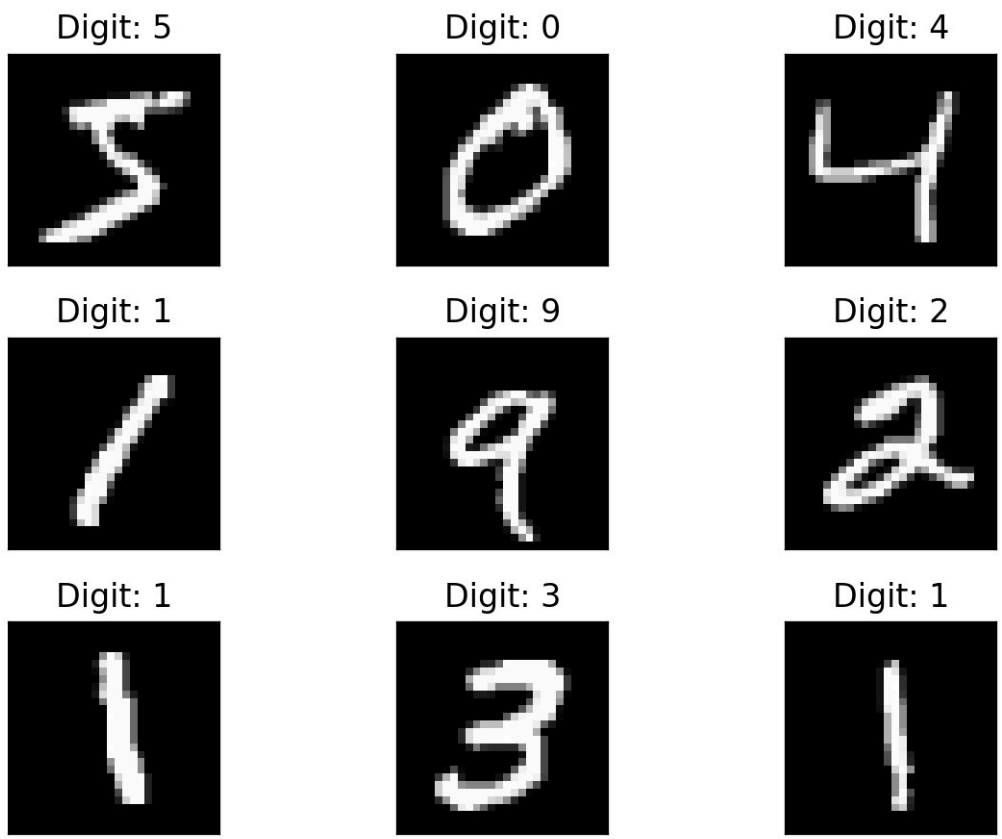
MNIST http://yann.lecun.com/exdb/mnist/

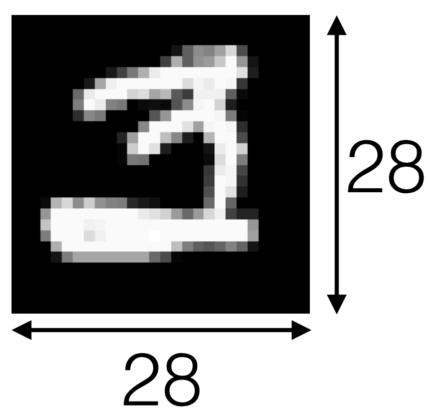
see Homework3.ipynb

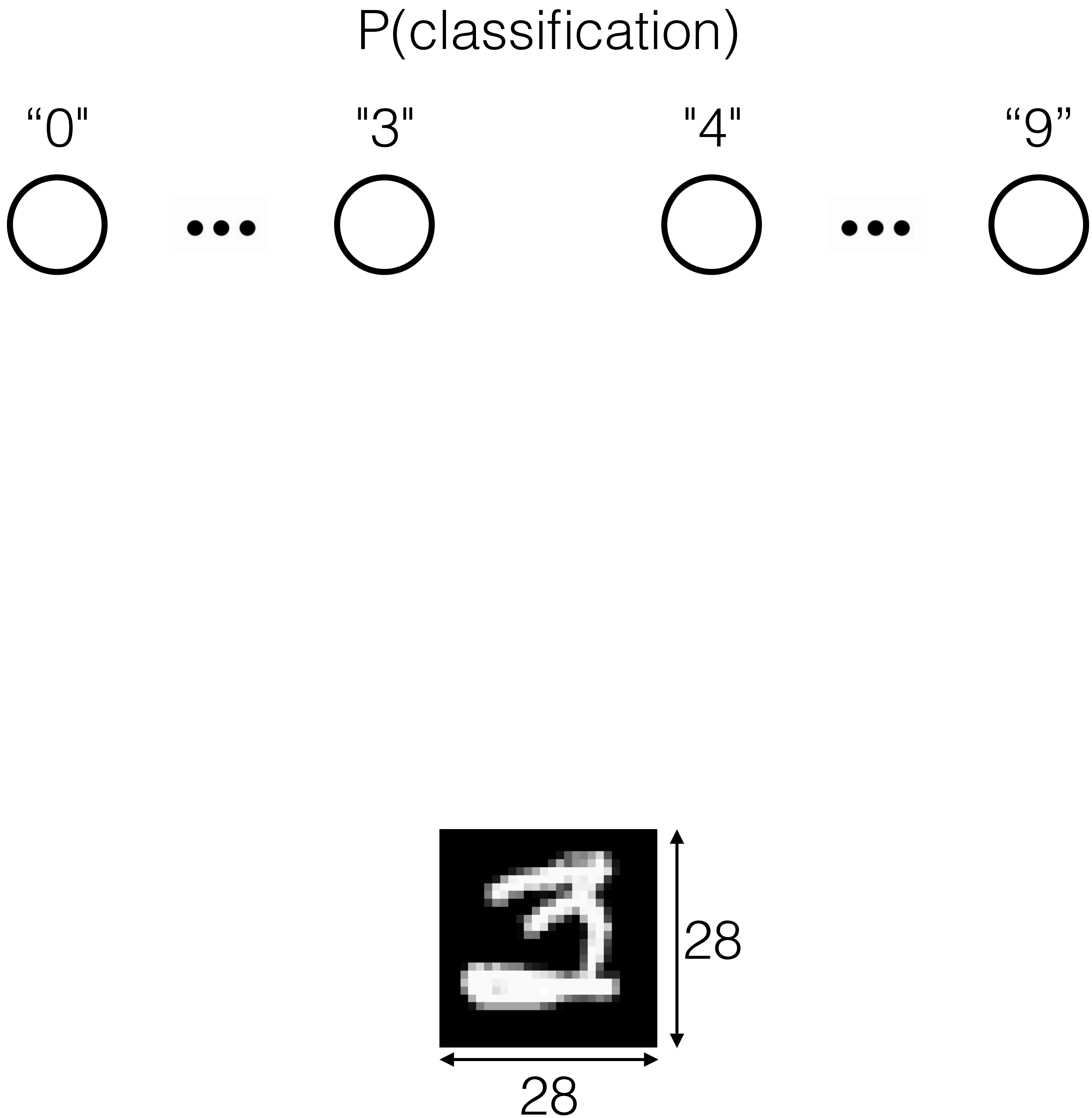
```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# check out dimensions and types of mnist data
print('Training images shape: ', train_images.shape)
print('Training images type:  ', type(train_images[0][0][0]))
print('Testing images shape:  ', test_images.shape)
print('Testing images type:   ', type(test_images[0][0][0]))
```

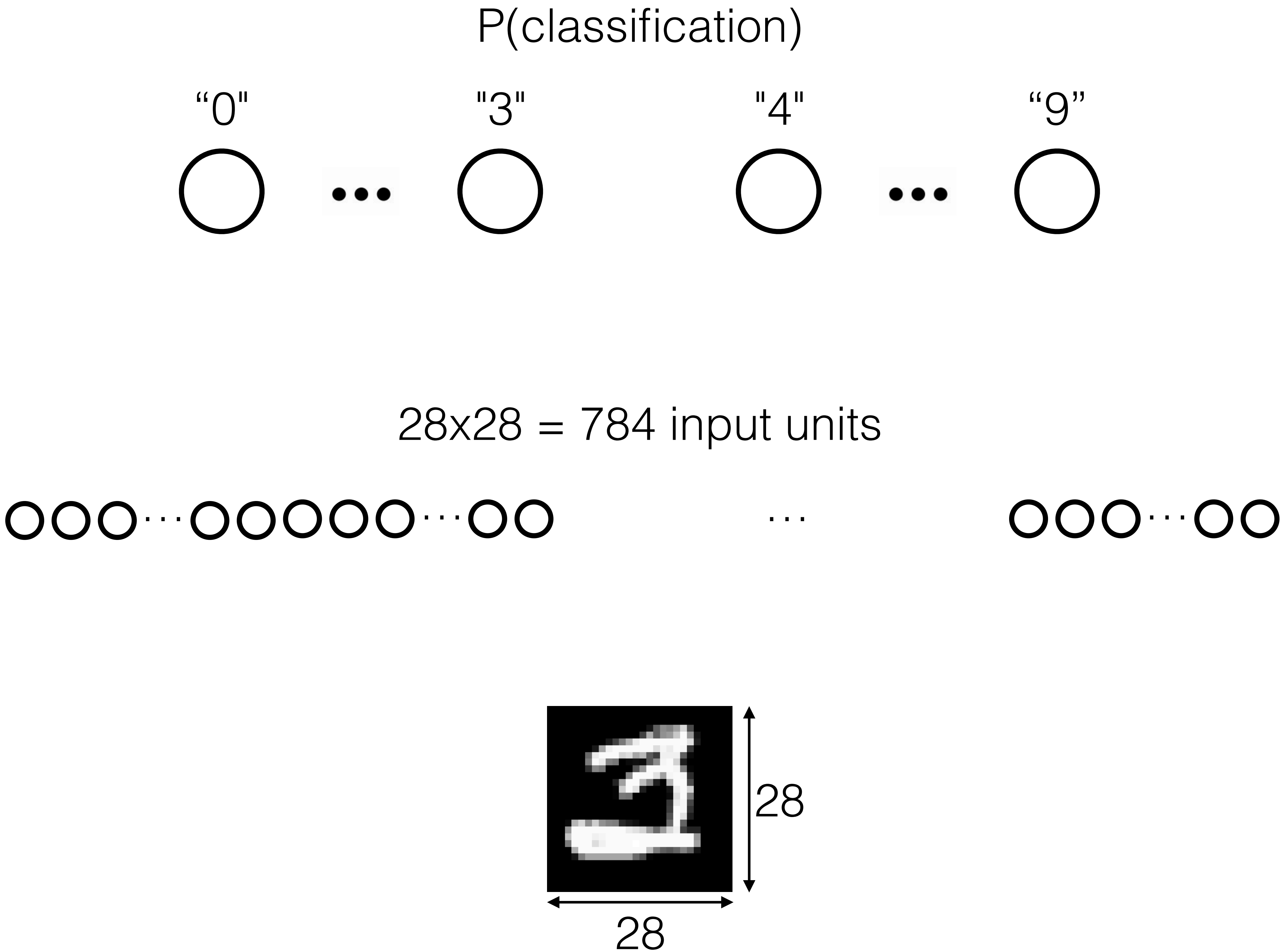
```
Training images shape:  (60000, 28, 28)
Training images type:   <class ‘numpy.uint8'>
Testing images shape:   (10000, 28, 28)
Testing images type:    <class 'numpy.uint8'>
```

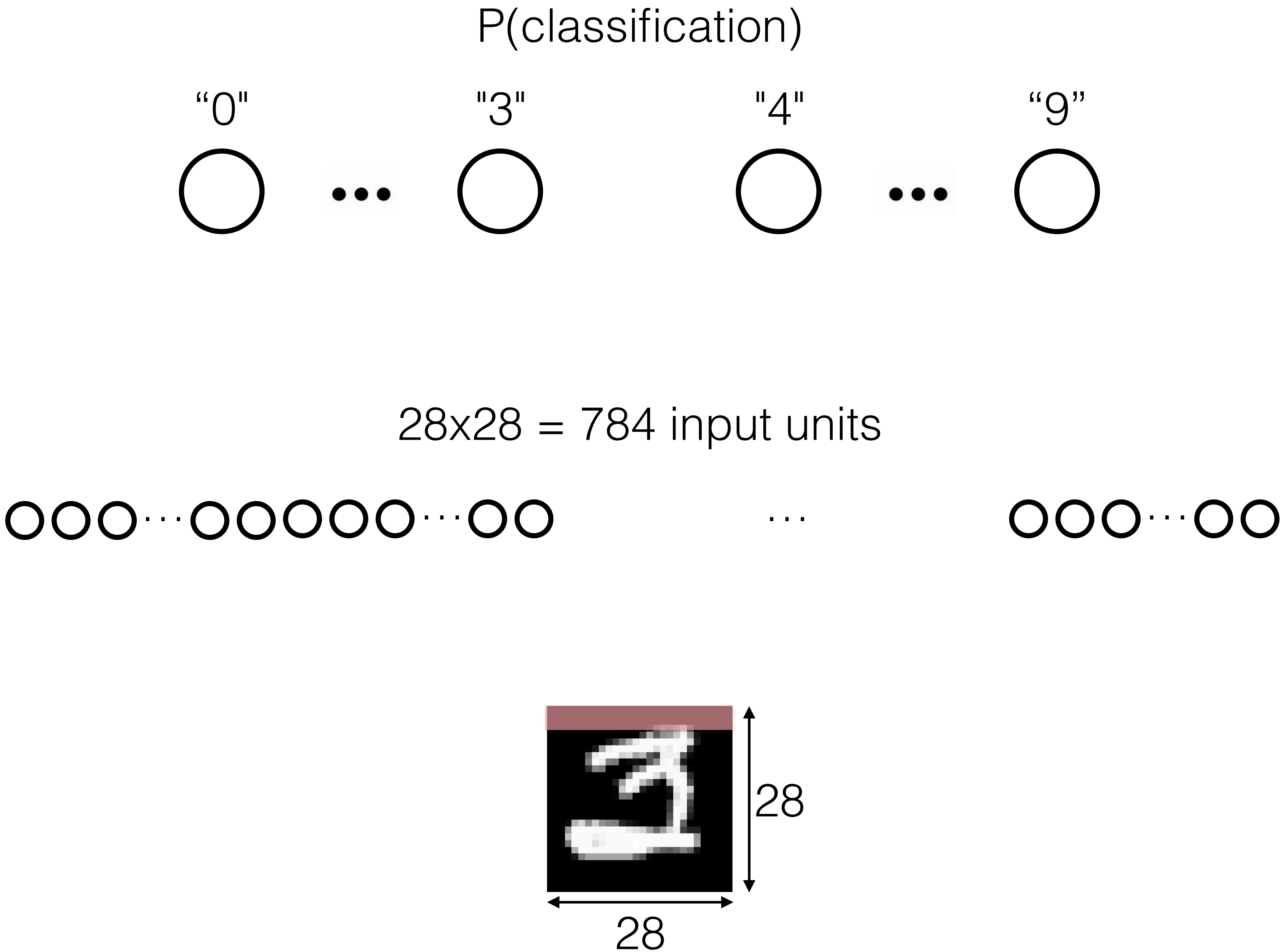
```
import matplotlib.pyplot as plt

# display some digits
fig = plt.figure()
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.tight_layout()
    plt.imshow(train_images[i], cmap='gray', interpolation='none')
    plt.title("Digit: {}".format(train_labels[i]))
    plt.xticks([])
    plt.yticks([])
plt.show()
```

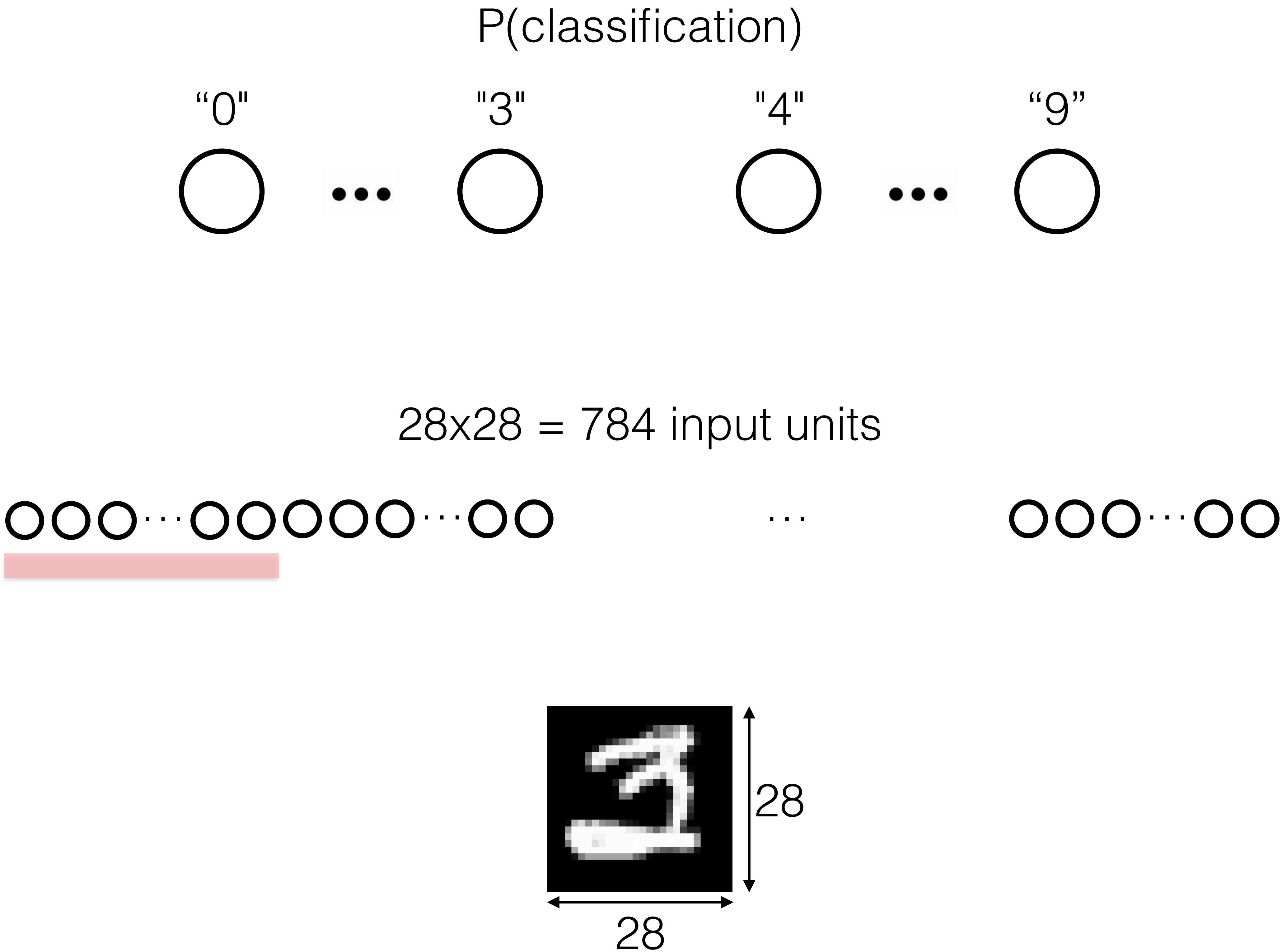


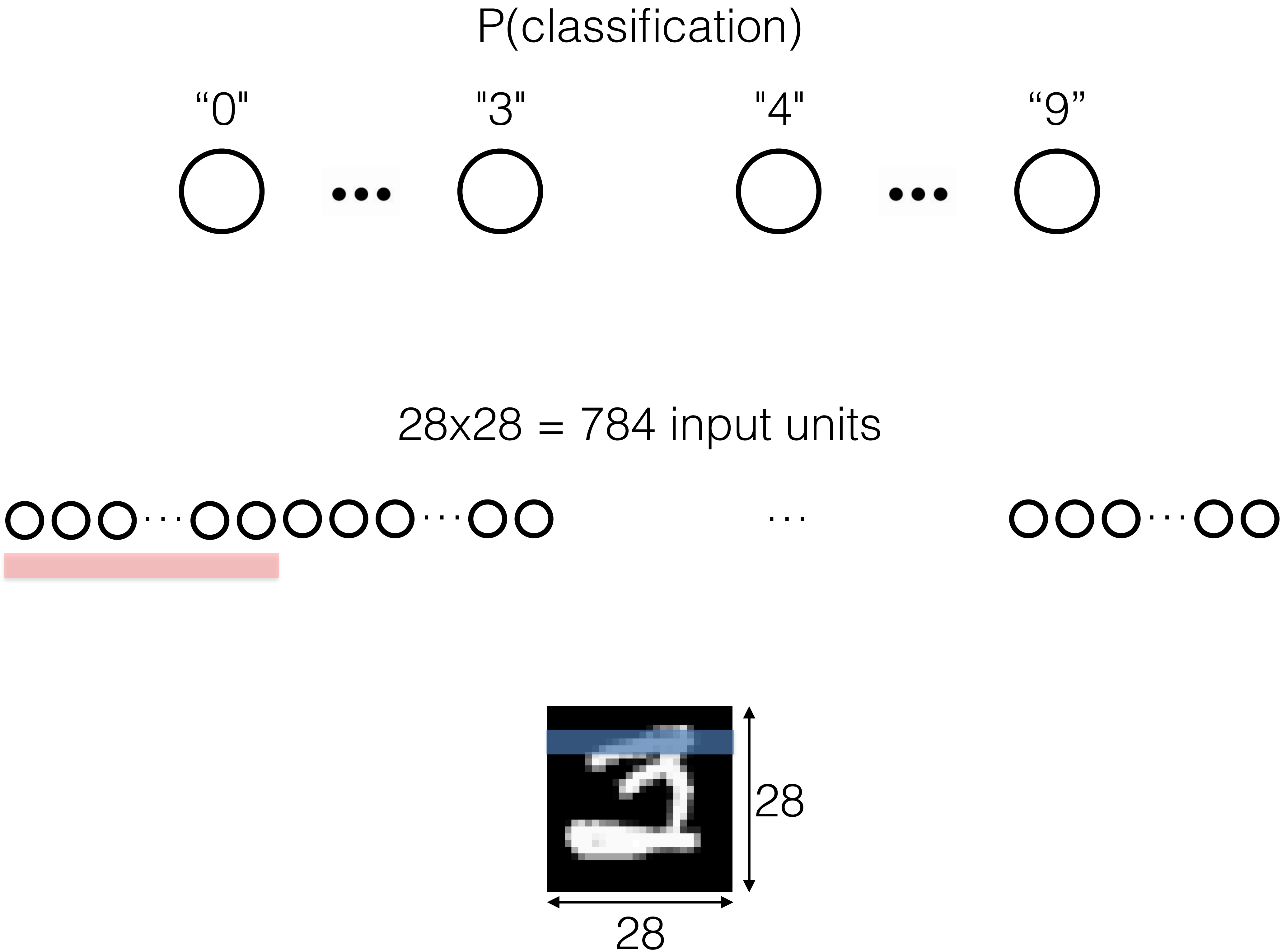

28
28


P(classification)
“0"
"3"
"4"
“9”
28
28

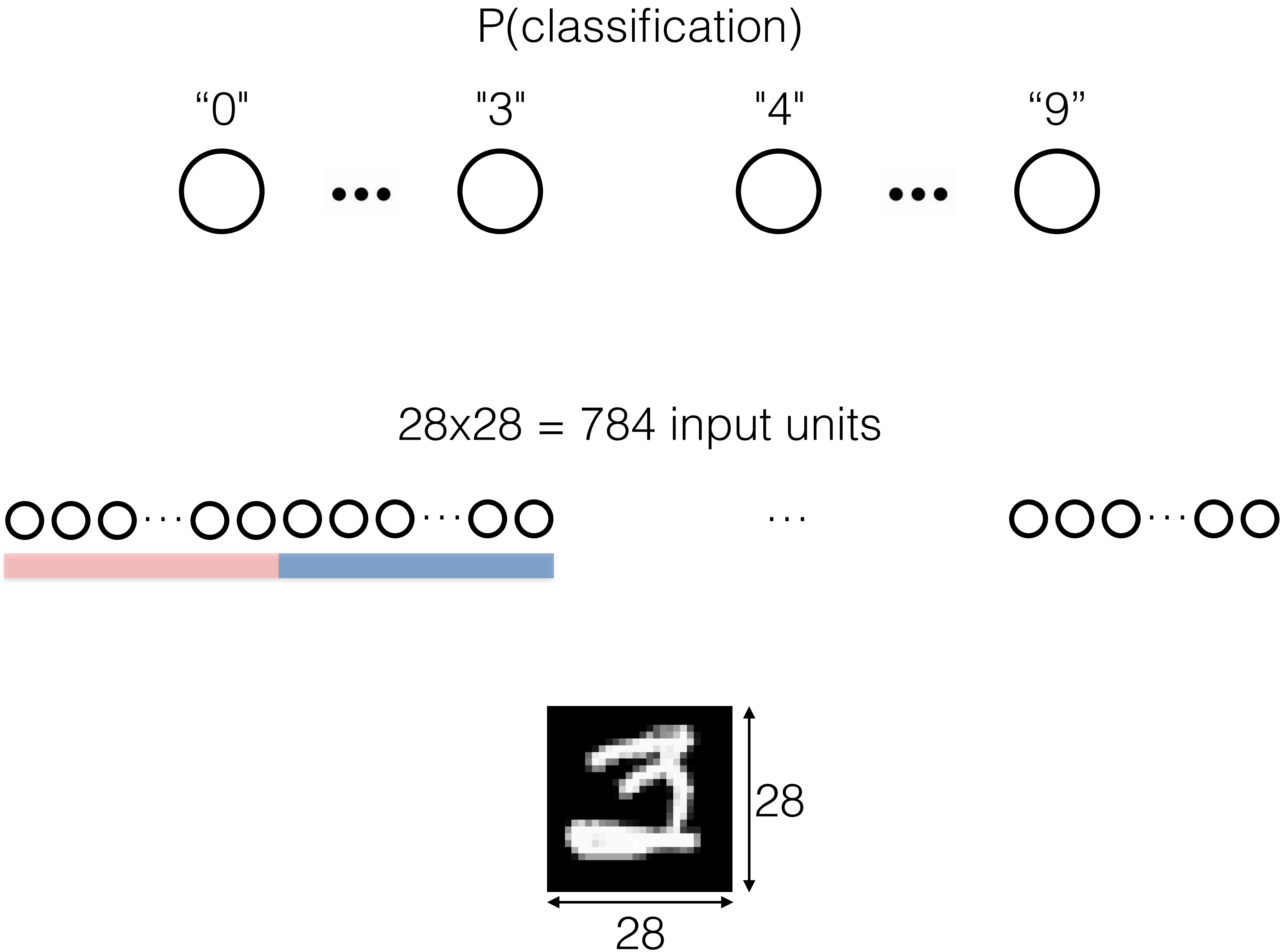

P(classification)
“0"
"3"
"4"
“9”
28x28 = 784 input units
28
28

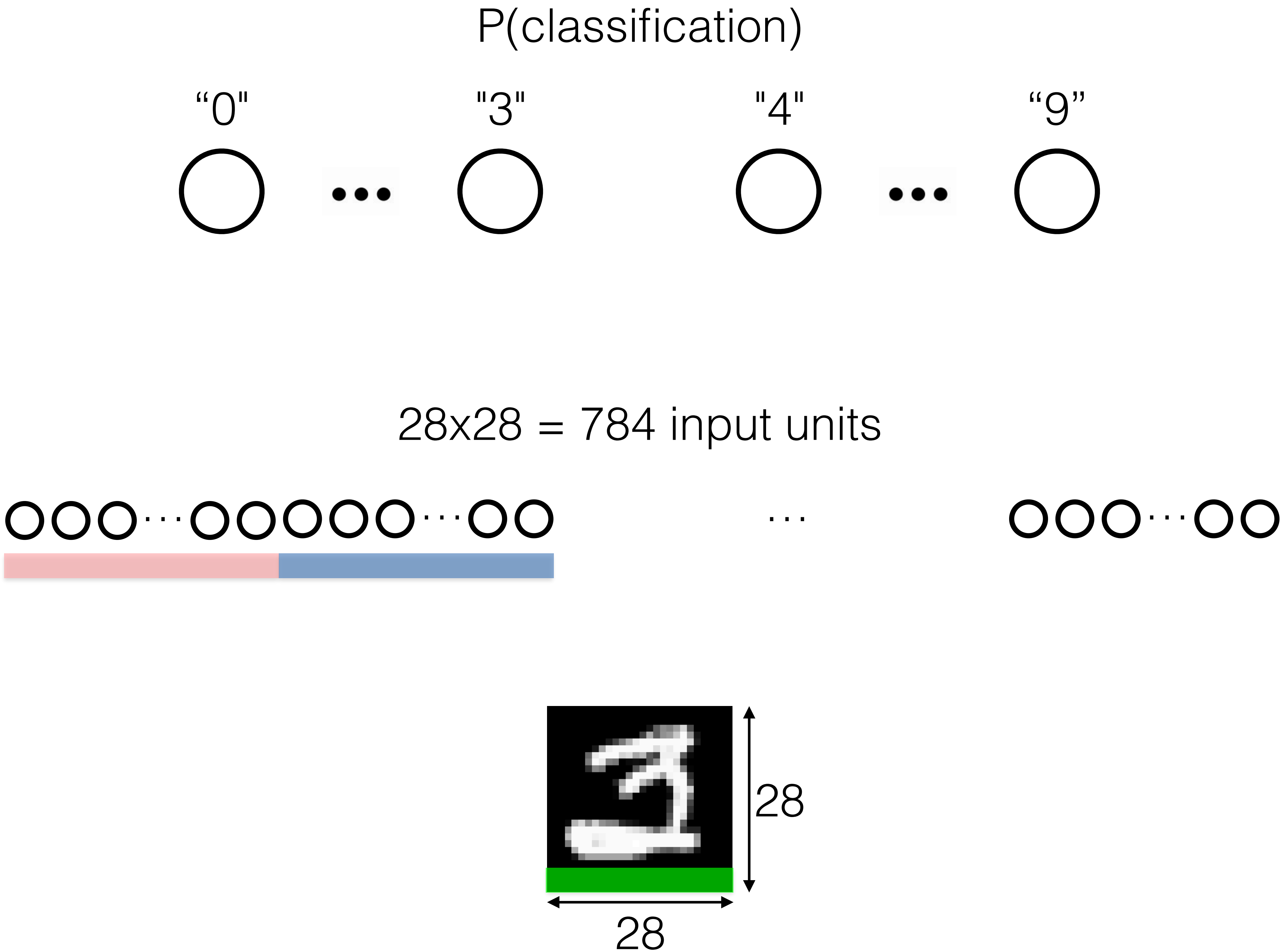

P(classification)
“0"
"3"
"4"
“9”
28x28 = 784 input units
28
28

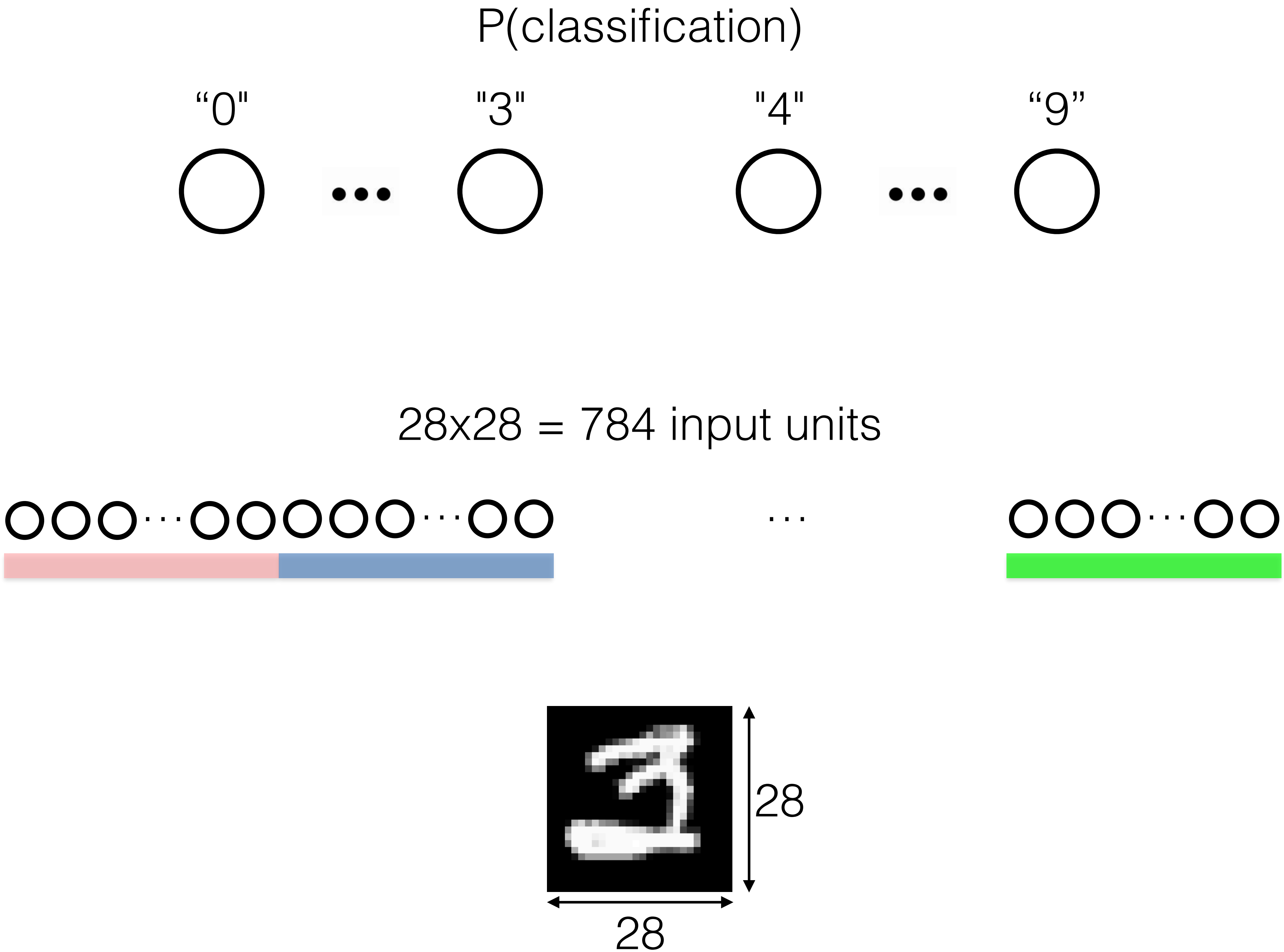

P(classification)
“0"
"3"
"4"
“9”
28x28 = 784 input units
28
28



P(classification)
“0"
"3"
"4"
“9”
28x28 = 784 input units
28
28

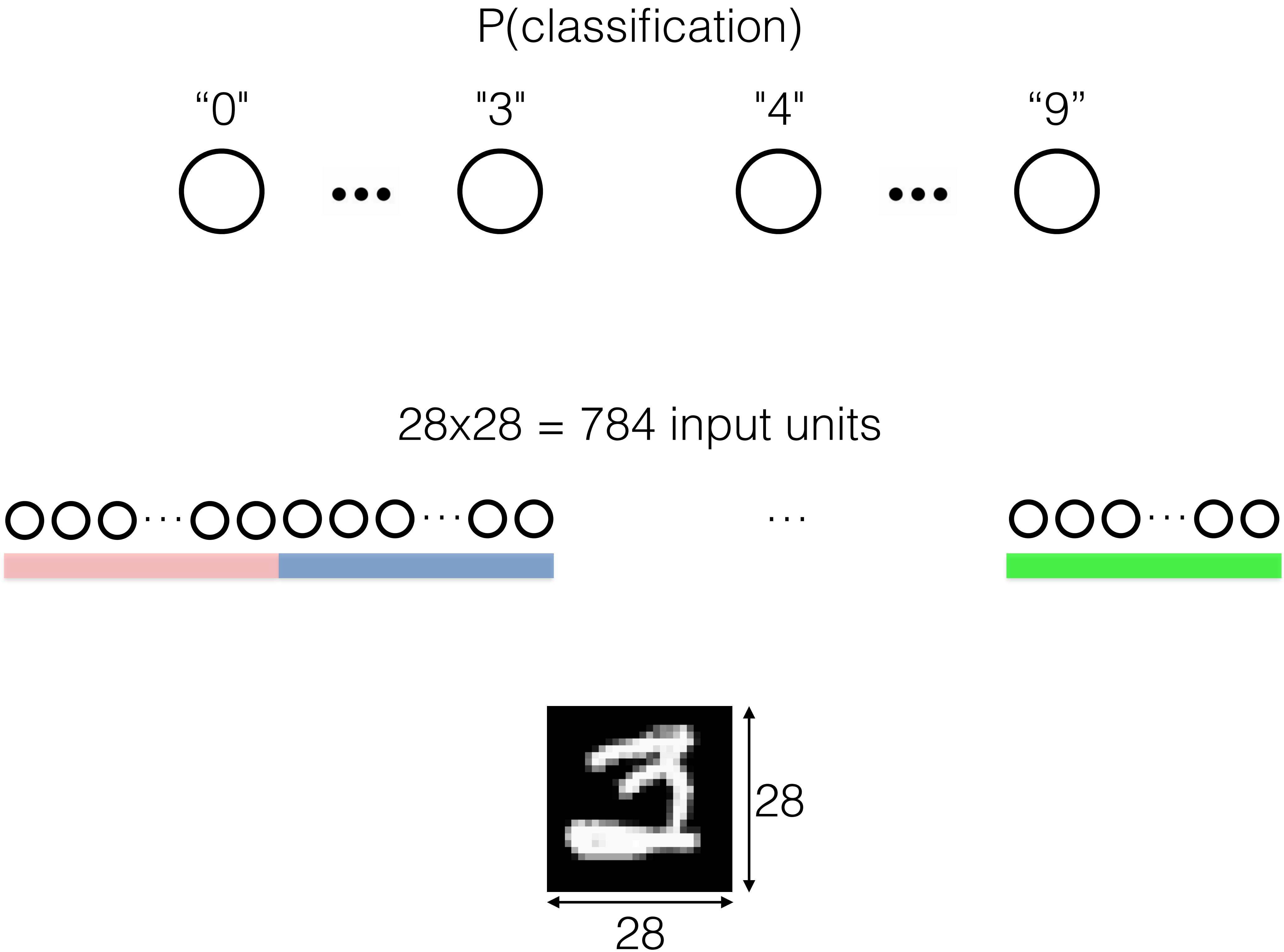

P(classification)
“0"
"3"
"4"
“9”
28x28 = 784 input units
28
28


P(classification)
“0"
"3"
"4"
“9”
28x28 = 784 input units
28
28


P(classification)
“0"
"3"
"4"
“9”
28x28 = 784 input units
28
28

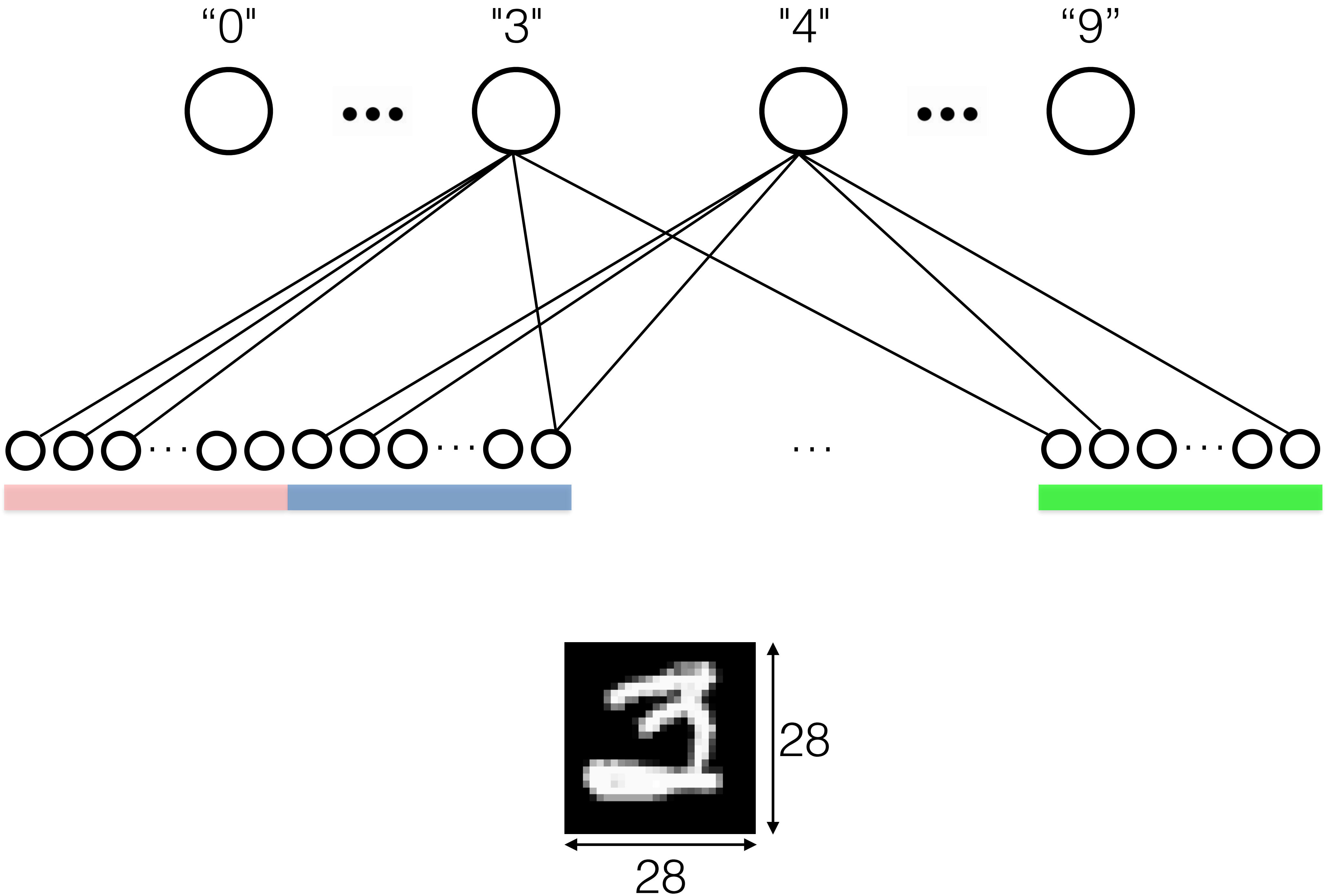
```
# need to reshape and preprocess the training/testing images
sz = train_images.shape[1]
train_images_vec = train_images.reshape((train_images.shape[0], sz * sz))
train_images_vec = train_images_vec.astype('float32') / 255
test_images_vec = test_images.reshape((test_images.shape[0], sz * sz))
test_images_vec = test_images_vec.astype(‘float32') / 255

# display new input dimensions/type
print('Training images shape: ', train_images_vec.shape)
print('Training images type:  ', type(train_images_vec[0][0]))
```

```
Training images shape:  (60000, 784)
Training images type:   <class ‘numpy.float32'>
Testing images shape:   (10000, 784)
Testing images type:    <class 'numpy.float32'>
```


P(classification)
“0"
"3"
"4"
“9”
28x28 = 784 input units
28
28

```
print('Training labels shape: ', train_labels.shape)
print('Training labels type:  ', type(train_labels[0]))

# also need to categorically encode the labels
print("First 5 training labels as labels:\n", train_labels[:5])
from keras.utils import to_categorical
train_labels_onehot = to_categorical(train_labels)
test_labels_onehot = to_categorical(test_labels)
print("First 5 training labels as one-hot encoded vectors:\n",
                       train_labels_onehot[:5])

# display new output dimensions/type
print('Training labels shape: ', train_labels_onehot.shape)
print('Training labels type:  ', type(train_labels_onehot[0][0]))
```

```
Training labels shape:  (60000,)
Training labels type:   <class ‘numpy.uint8'>

First 5 training labels as labels:
 [5 0 4 1 9]
First 5 training labels as one-hot encoded vectors:
 [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]

Training labels shape:  (60000, 10)
Training labels type:   <class 'numpy.float32'>
```

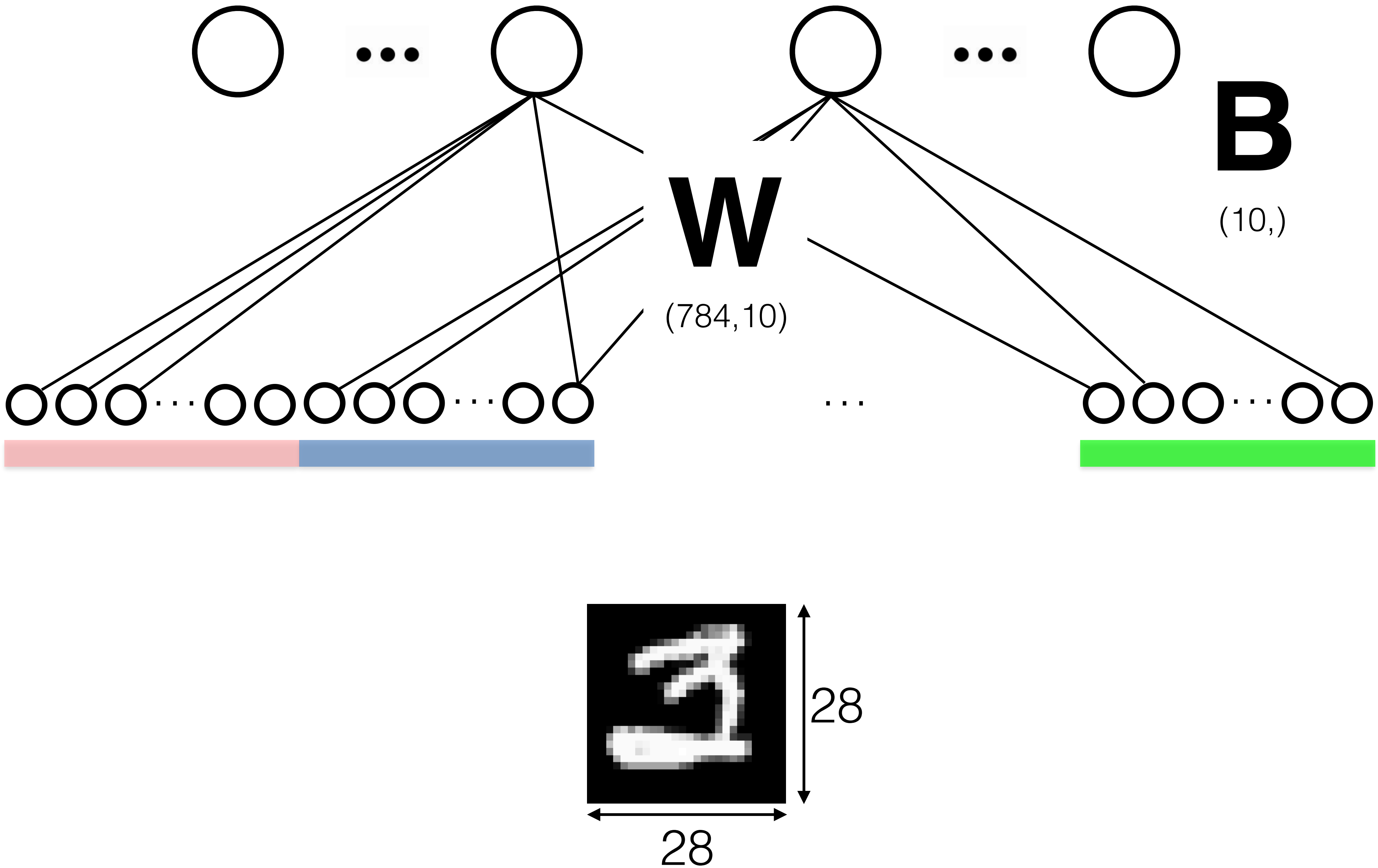


“0"
"3"
"4"
“9”
28
28

```
# import tools for basic keras networks
from keras import models
from keras import layers

nout = 10
# create architecture of simple neural network model
# input layer  : 28*28 = 784 input nodes
# output layer : 10 (nout) output nodes
network = models.Sequential()
network.add(layers.Dense(nout, activation='sigmoid', input_shape=(sz * sz,)))

# compile network
network.compile(optimizer='sgd', loss='mean_squared_error', metrics=['accuracy'])

# now train the network
history = network.fit(train_images_vec, train_labels_onehot, verbose=False,
                       validation_split=.1, epochs=20, batch_size=128)
```

$$a_j = \frac{1}{1+\exp(-n_j)}$$

the weights (**W**) and biases (**B**) are learned from training data



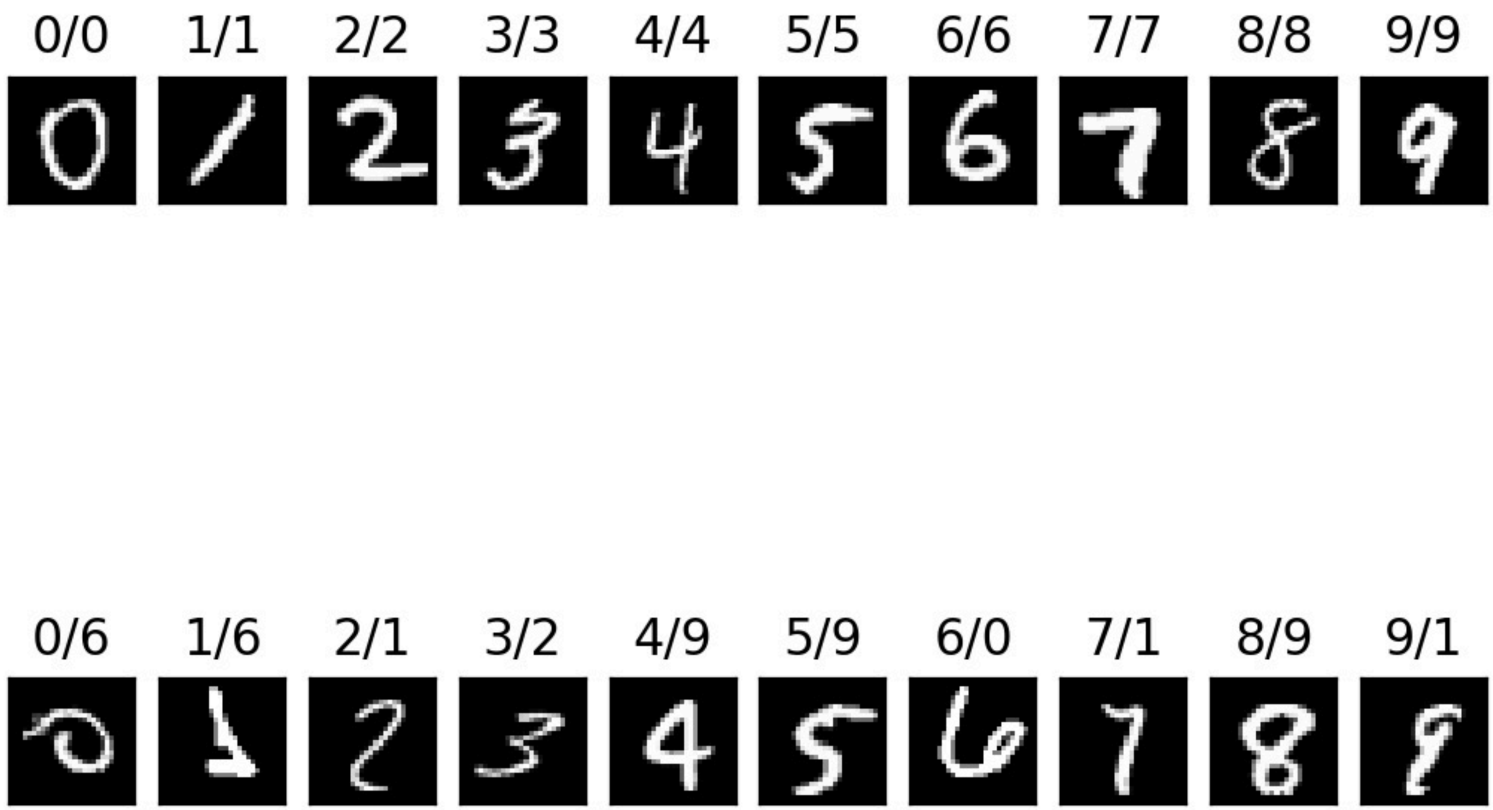
Homework 3

see Homework3.py and Homework3.ipynb
on Brightspace

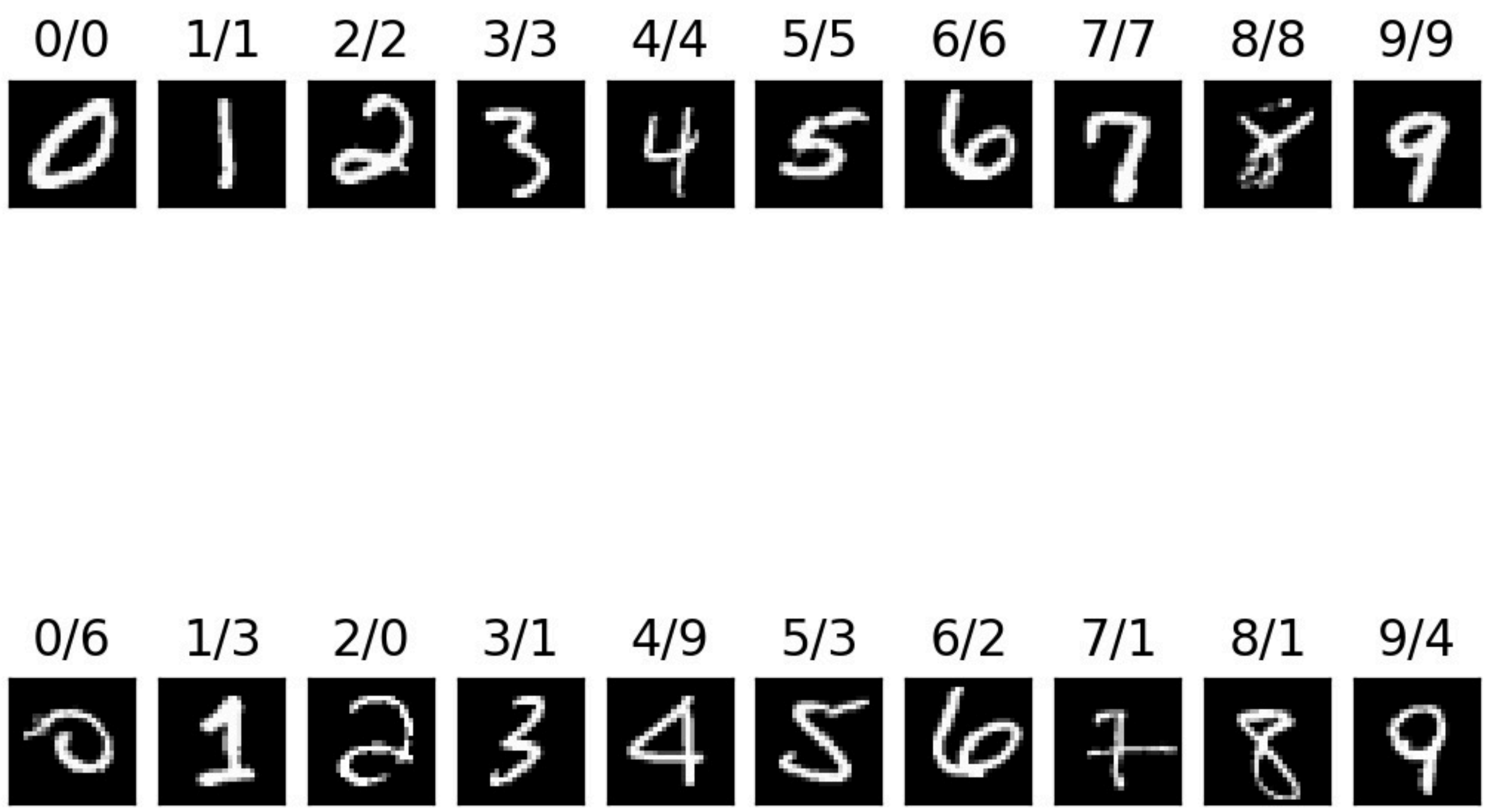
20 points
Due Thursday January 24th

```
## Q1. The original MNIST test_labels numpy array contains the digit value associated
## with the corresponding digit image (test_images). The output from the network (from
## out = network.predict(test_images_vec) above) contains the activations of the 10
## output nodes for every test image presented to the network. Write a function that
## takes the (10000,10) numpy array of output activations (of type float32) and returns
## a (10000,) numpy array of discrete digit classification by the network (of type uint8).
## In other words, create a test_decisions numpy array of the same size and type as the
## MNIST test_labels array you started with. Below you will use both arrays to pull out
## test images that the network classifies correctly or incorrectly.
##
## To turn a numpy array of continuous output activations into a discrete digit classification,
## just take the maximum output as the "winner" that take all, determining the classification.
##
## In your function, feel free to use for loops. We are looking to see that you understand
## how to use the outputs generated by the network, not whether you can program using the
## most efficient python style.
```

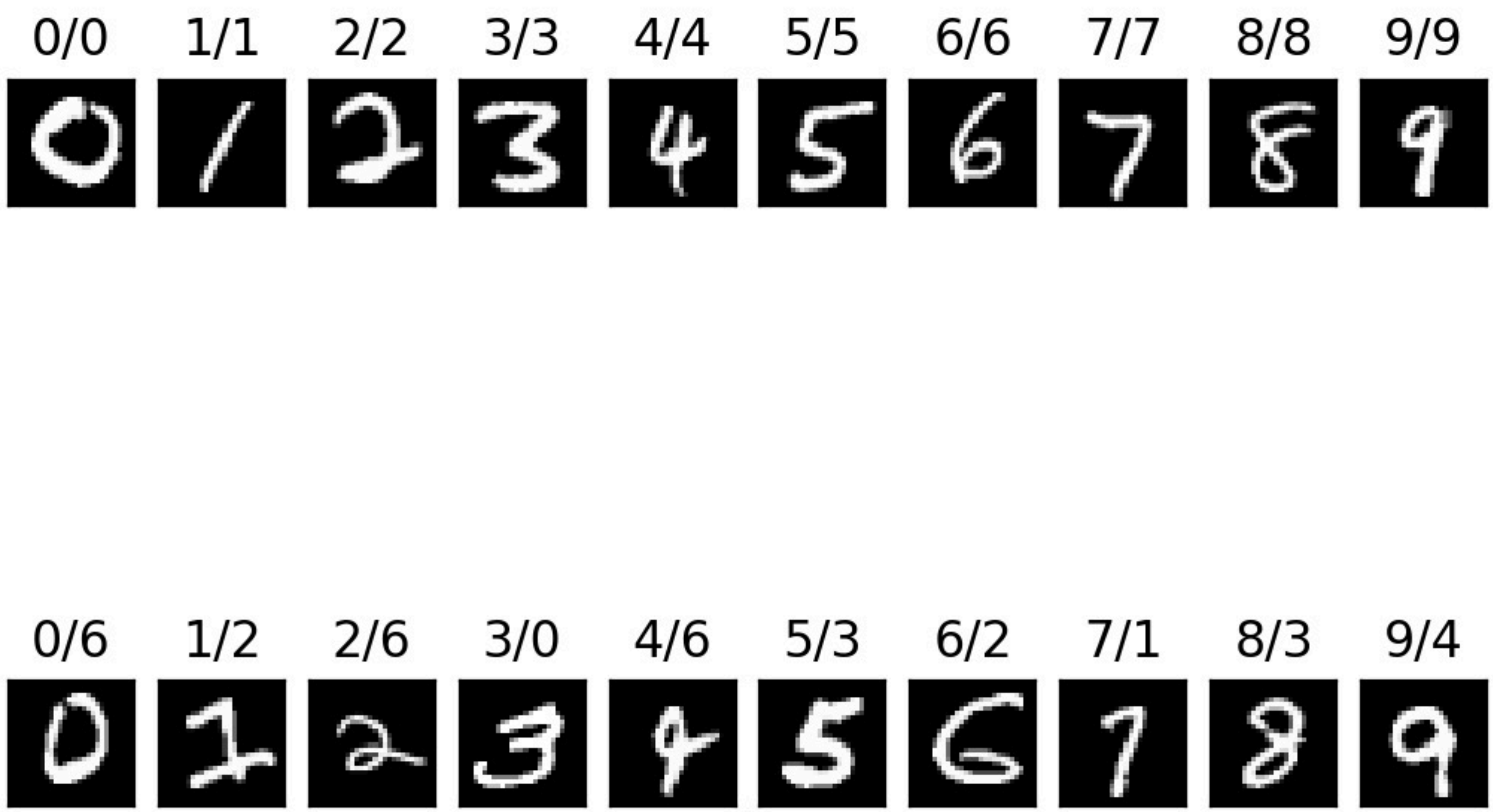
Q2. Comparing the correct answers (test_labels) and network classifications
(test_decisions), for each digit 0..9, find one test image (test_image) that is classified
by the network correctly and one test image that is classified by the network incorrectly.
##
Create a 2x10 plot of digit images (feel free to adapt the code above that uses subplot),
with a column for each digit 0..9 with the first row showing examples correctly classified
(one example for each digit) and the second row showing the examples incorrectly
classified (one example for each digit). Each subplot title should show the answer and
the classification response (e.g., displaying 4/2 as the title, if the correct answer is 4
and the classification was 2).



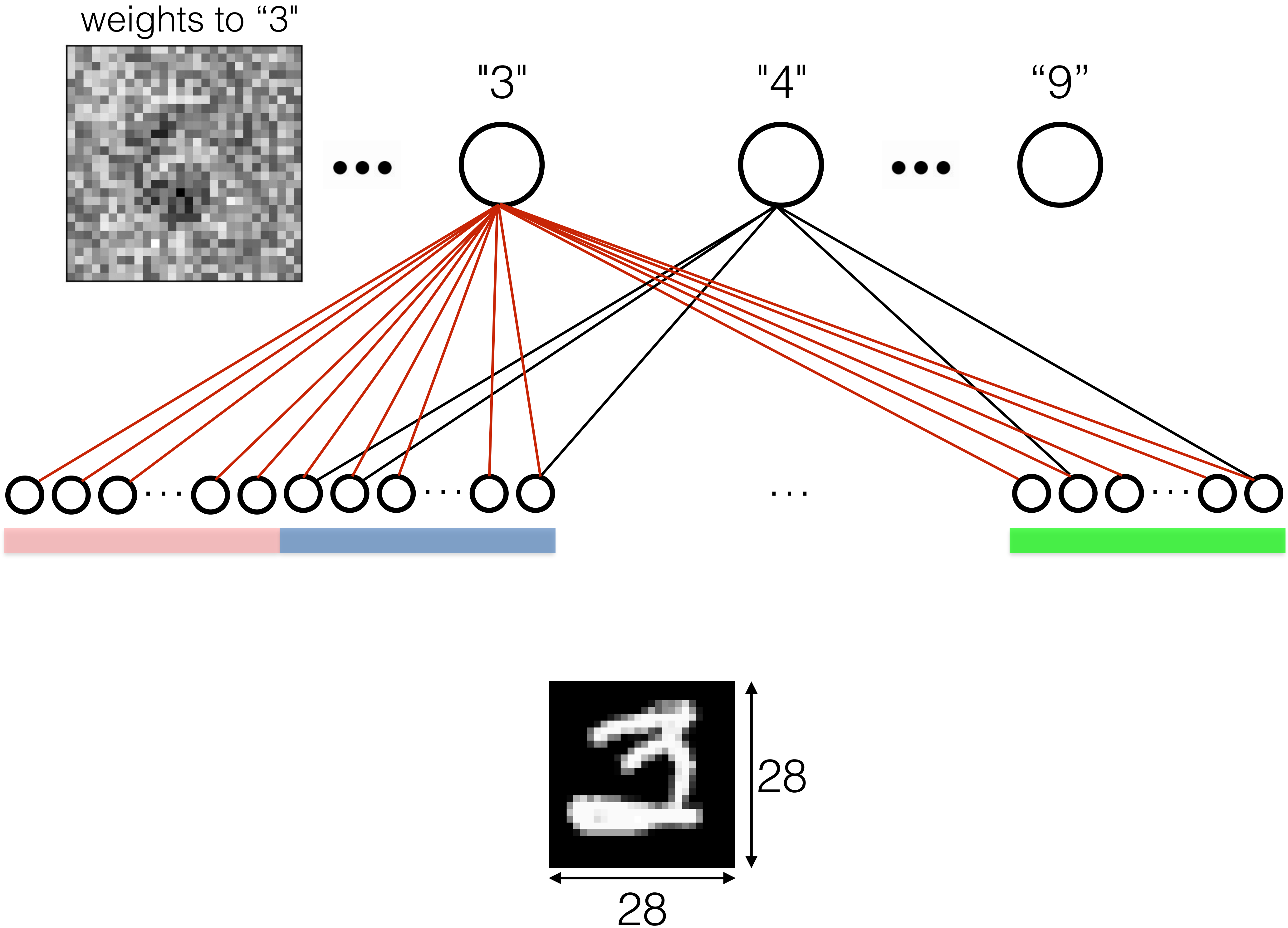
Q2. Comparing the correct answers (test_labels) and network classifications
(test_decisions), for each digit 0..9, find one test image (test_image) that is classified
by the network correctly and one test image that is classified by the network incorrectly.
##
Create a 2x10 plot of digit images (feel free to adapt the code above that uses subplot),
with a column for each digit 0..9 with the first row showing examples correctly classified
(one example for each digit) and the second row showing the examples incorrectly
classified (one example for each digit). Each subplot title should show the answer and
the classification response (e.g., displaying 4/2 as the title, if the correct answer is 4
and the classification was 2).



Q2. Comparing the correct answers (test_labels) and network classifications
(test_decisions), for each digit 0..9, find one test image (test_image) that is classified
by the network correctly and one test image that is classified by the network incorrectly.
##
Create a 2x10 plot of digit images (feel free to adapt the code above that uses subplot),
with a column for each digit 0..9 with the first row showing examples correctly classified
(one example for each digit) and the second row showing the examples incorrectly
classified (one example for each digit). Each subplot title should show the answer and
the classification response (e.g., displaying 4/2 as the title, if the correct answer is 4
and the classification was 2).



Q3. Create "images" of the connection weight adapting the code used to display
the actual digit images. There should be 10 weight images, an image for each
set of weight connecting the input layer (784 inputs) to each output node.
You will want to reshape the (784,1) vector of weights to a (28,28) image and
display the result using imshow().



weights to “3"
"3"
"4"
“9”
28
28

Q4. Use the weight matrix (W), bias vector (B), and activation function (simple sigmoid)
to reproduce in your own code the outputs (out) generated by the network (from
this out = network.predict(test_images_vec))
##
The simple sigmoid activation function is defined as follows:
f(x) = 1 / (1+exp(-x))
##
Feel free to use for loops or vector/matrix operations (we will go over the latter in
in the coming weeks)
##
Confirm that your output vectors and the keras-produced output vectors are the same
(within some small epsilon since floating point calculations will often not come out
exactly the same on computers).