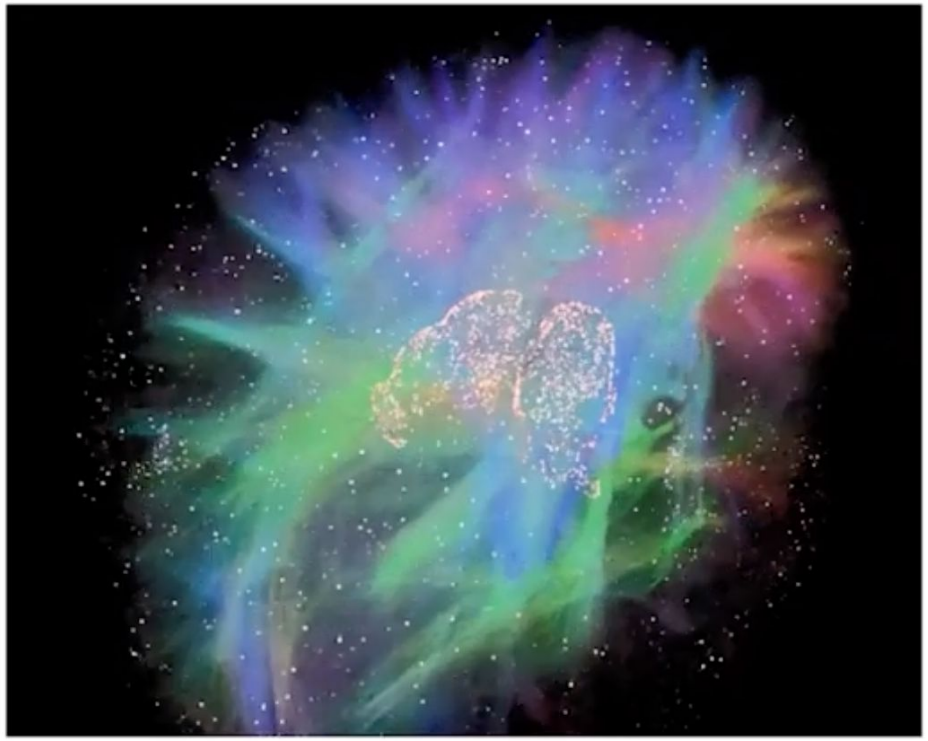
Deep Learning: Session 4

Deep Neural Nets

Outline

1. Recap
2. Deep L-Layers
3. Divide and Conquer
4. The Neuron: Pattern Recognition
5. DNNs

In previous chapters...



Human Brain

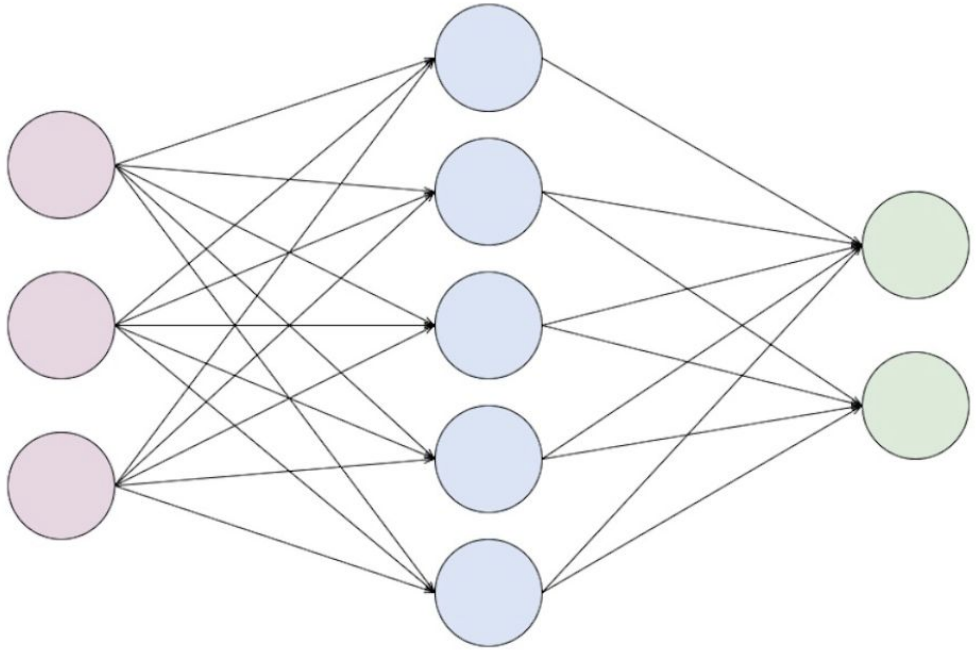
- Thalamocortical system:
  3 million neurons
  476 million synapses
- Full brain:
  100 billion neurons
  1,000 trillion synapses

Artificial Neural Network

- ResNet-152:
  60 million synapses

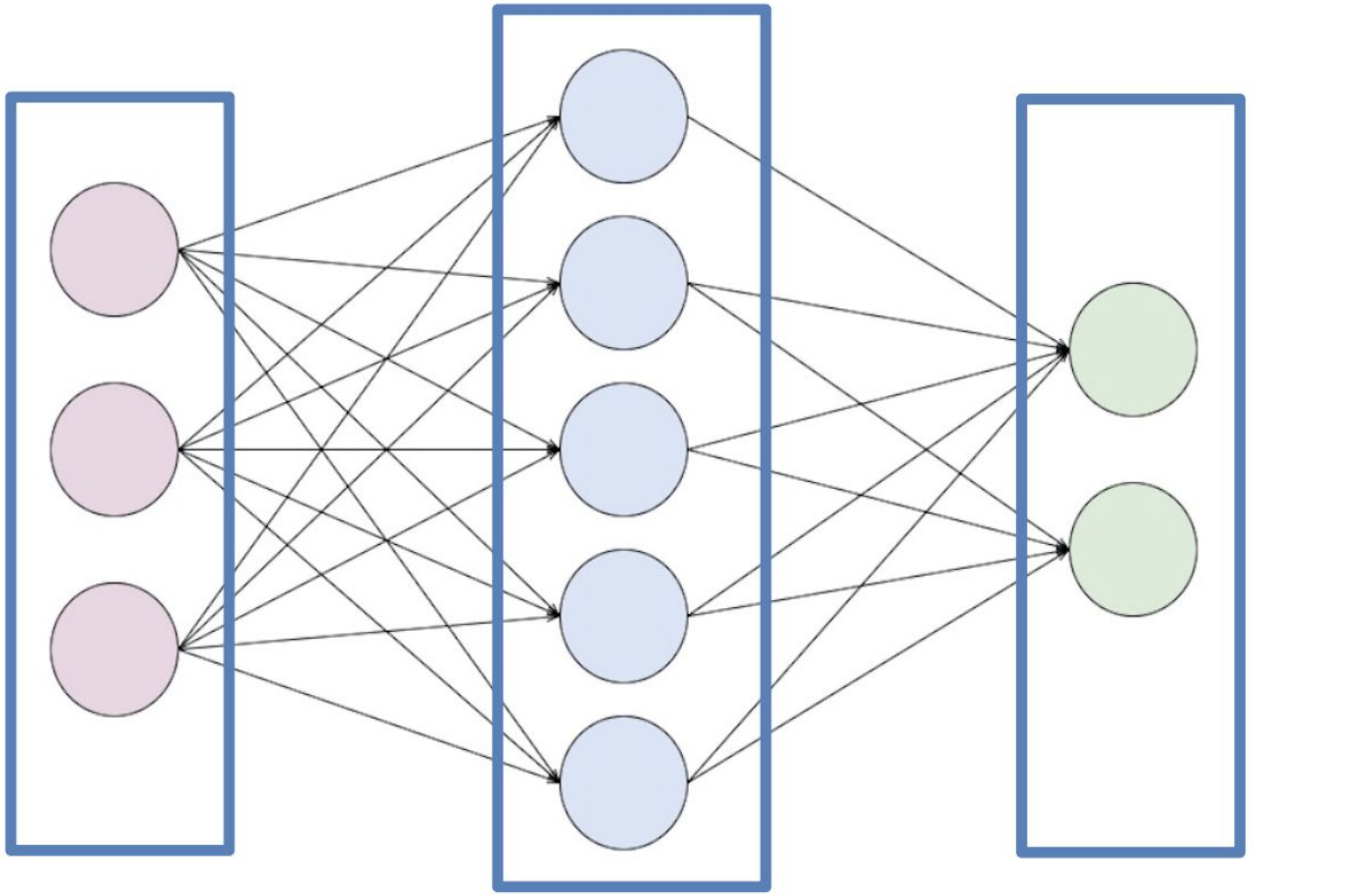
Human brains have ~10,000,000 times synapses
than artificial neural networks.

In previous chapters...



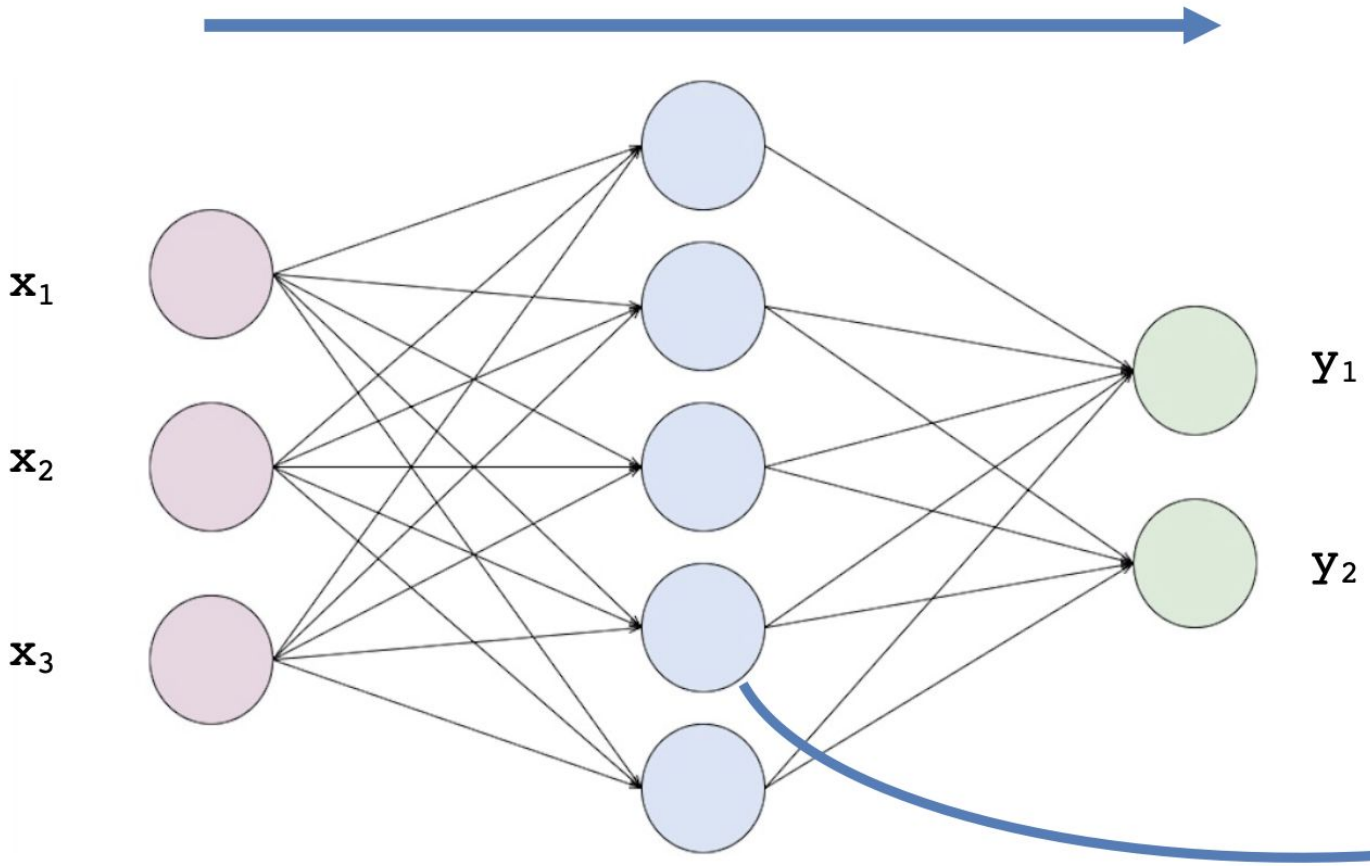
A net of information

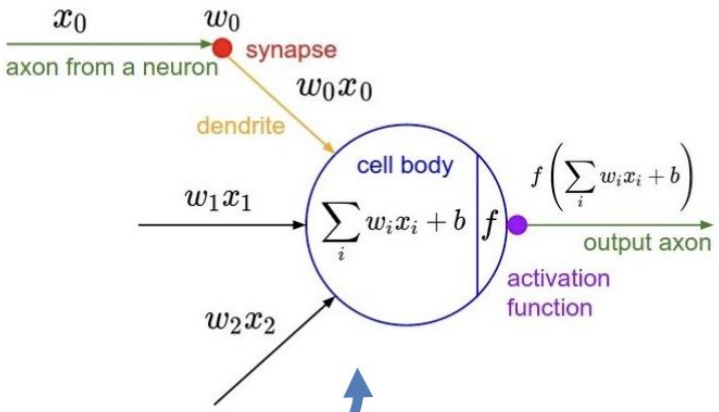
In previous chapters...



In previous chapters...

$$Y = F(x)$$

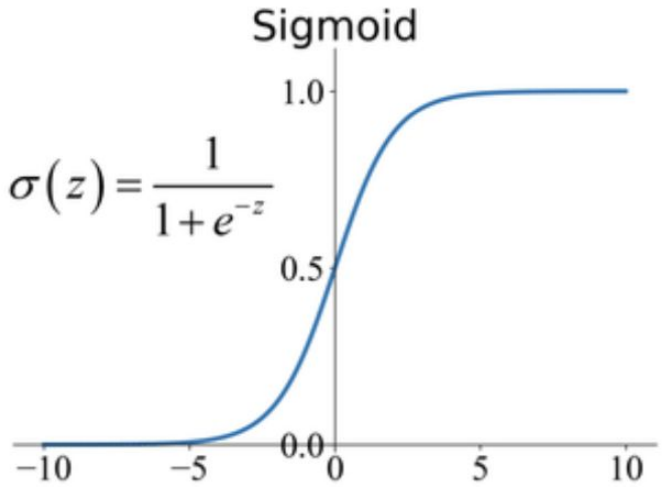




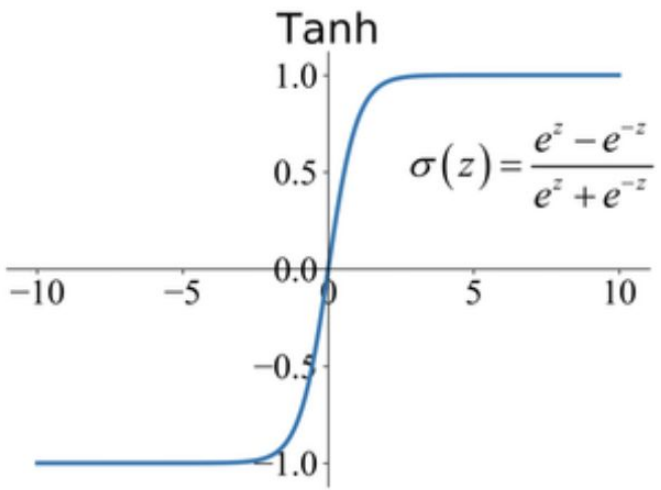
In previous chapters...

```
my_first_neural_net = tf.keras.Sequential([
    layers.Input(shape=(3, ), name="input"),
    layers.Dense(units=5, activation="sigmoid", name="hidden"),
    layers.Dense(units=2, activation="softmax", name="output")
])
```

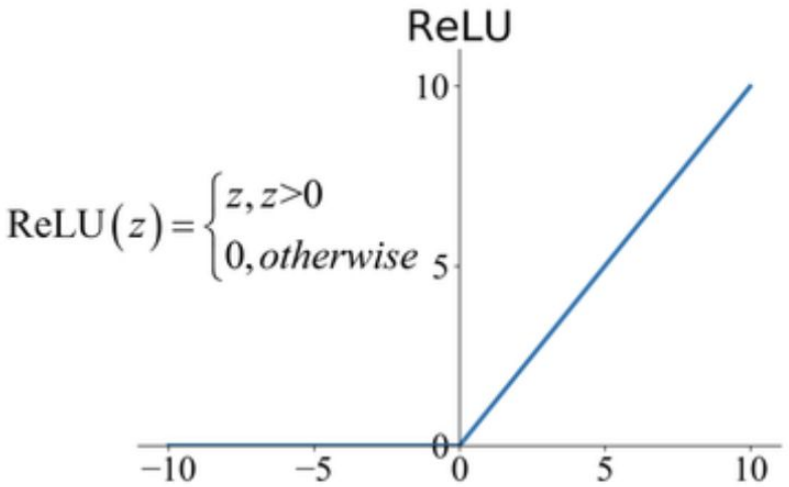
In previous chapters...



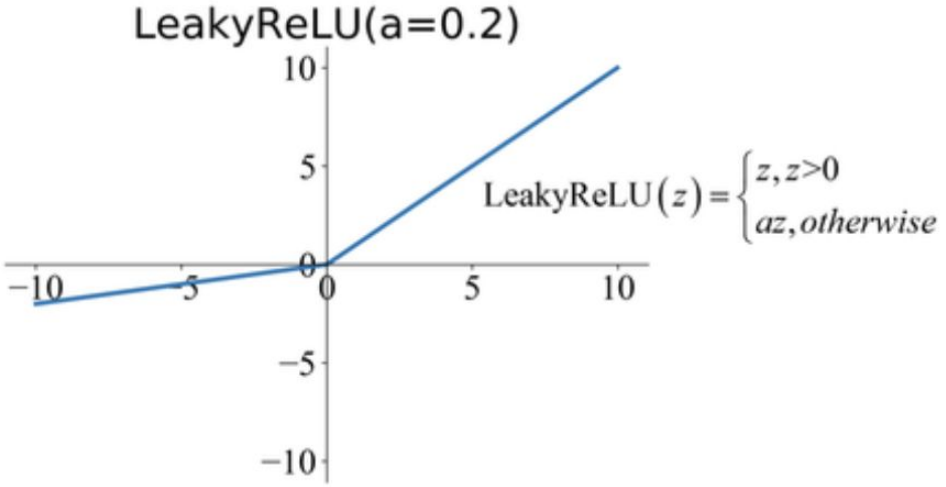
(a)



(b)



(c)



(d)

In previous chapters...

- The combination of linear functions is just another linear function. No matter the number of hidden layers, if we use linear functions as activations, we could just deal with outputs that are a linear combination of our inputs... not interesting.

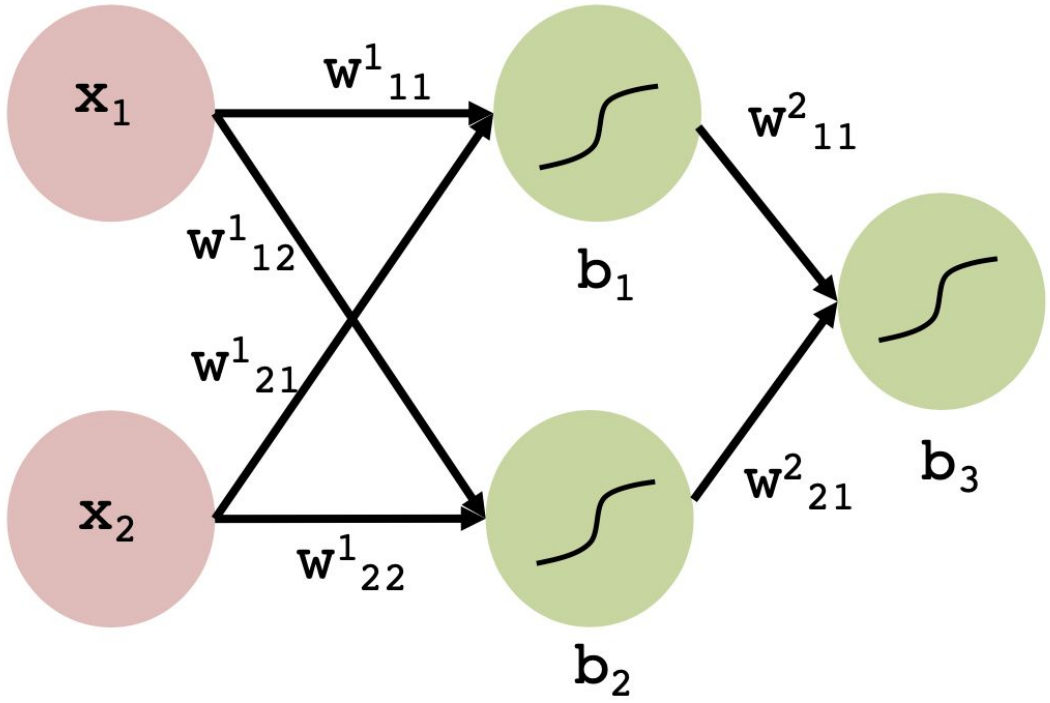
- Non-linear activations functions are key in neural nets to represent non-linear mapping functions.

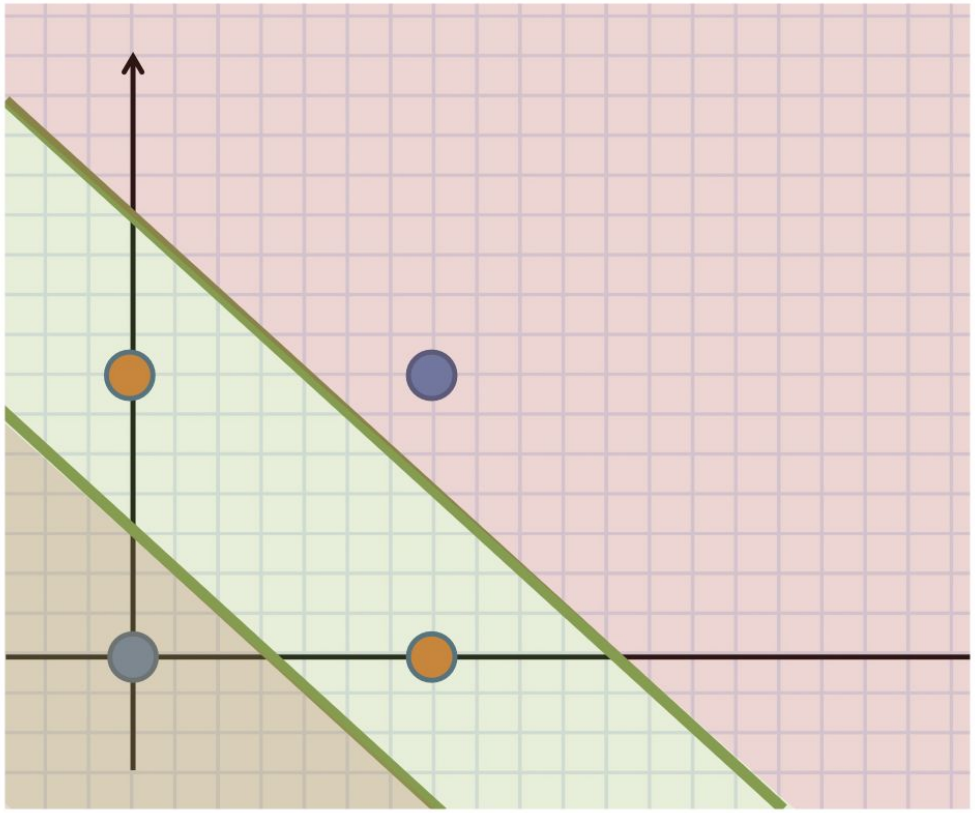
- **In the output layer**, at general: use sigmoid for binary classification, softmax for multiclass classification and linear for regression.

    - You can use softmax for binary classification using two outputs and in the case of a problem where data that can belong to multiple classes, then use sigmoid. [i.e., a picture of Audi belongs to the classes' car and vehicle]

- **In hidden layers:**
  - Tanh use to work better than sigmoid as it centers data.
  - ReLu is faster and it works very well too. It is the most used.

In previous chapters...





$$y = s(s(w^1_{11}x_1 + w^1_{21}x_2 + b_1) + s(w^1_{21}x_1 + w^1_{22}x_2 + b_2) + b_3)$$

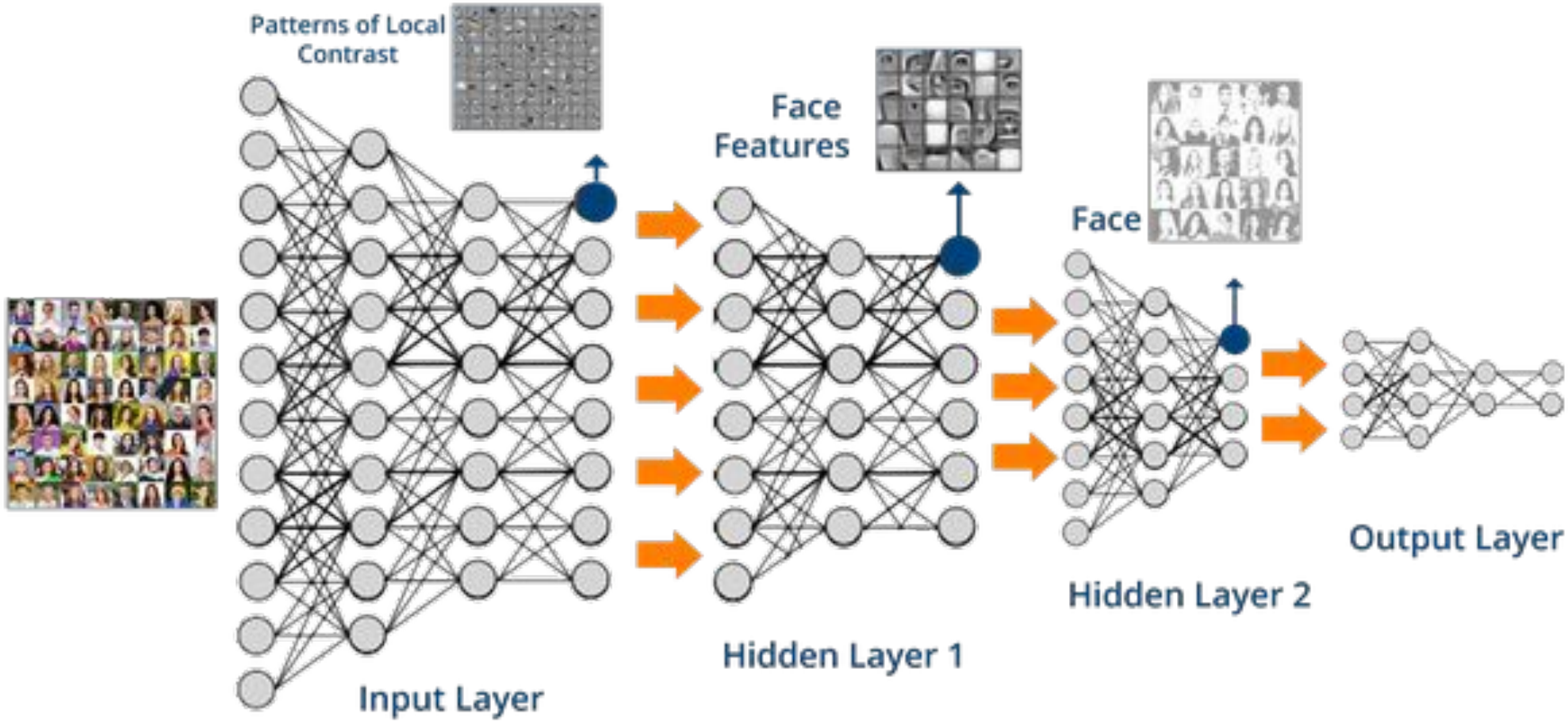
Outline

1. Recap
2. Deep L-Layers
3. Divide and Conquer
4. The Neuron: Pattern Recognition
5. DNNs

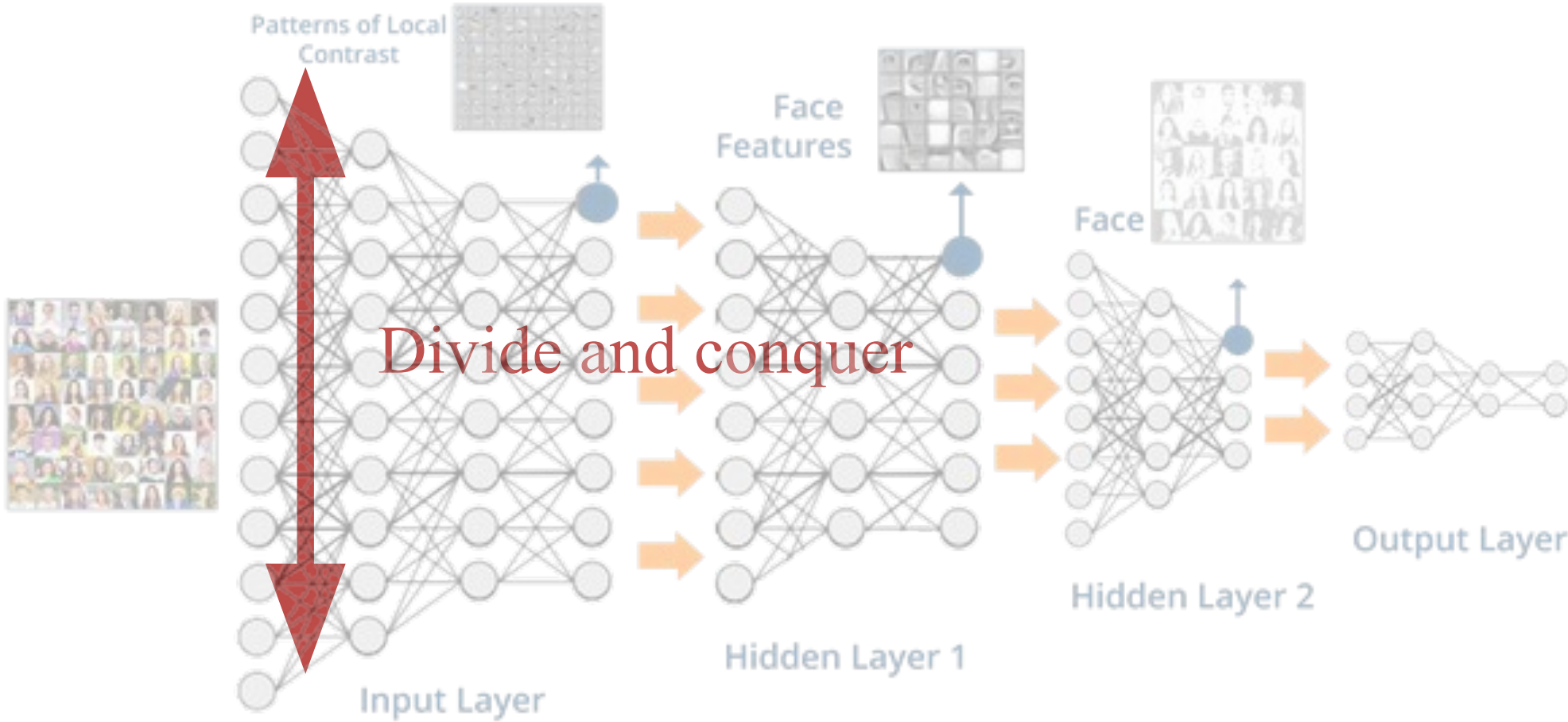
Why Deep L-layer Neural Nets

- From circuit theory:
  - We know that for representing some functions we could need many components in shallow architectures.
  - ... Functions that we can easily represent if we include more layers.
- The Divide & Conquer strategy is powerful, and it is efficient.

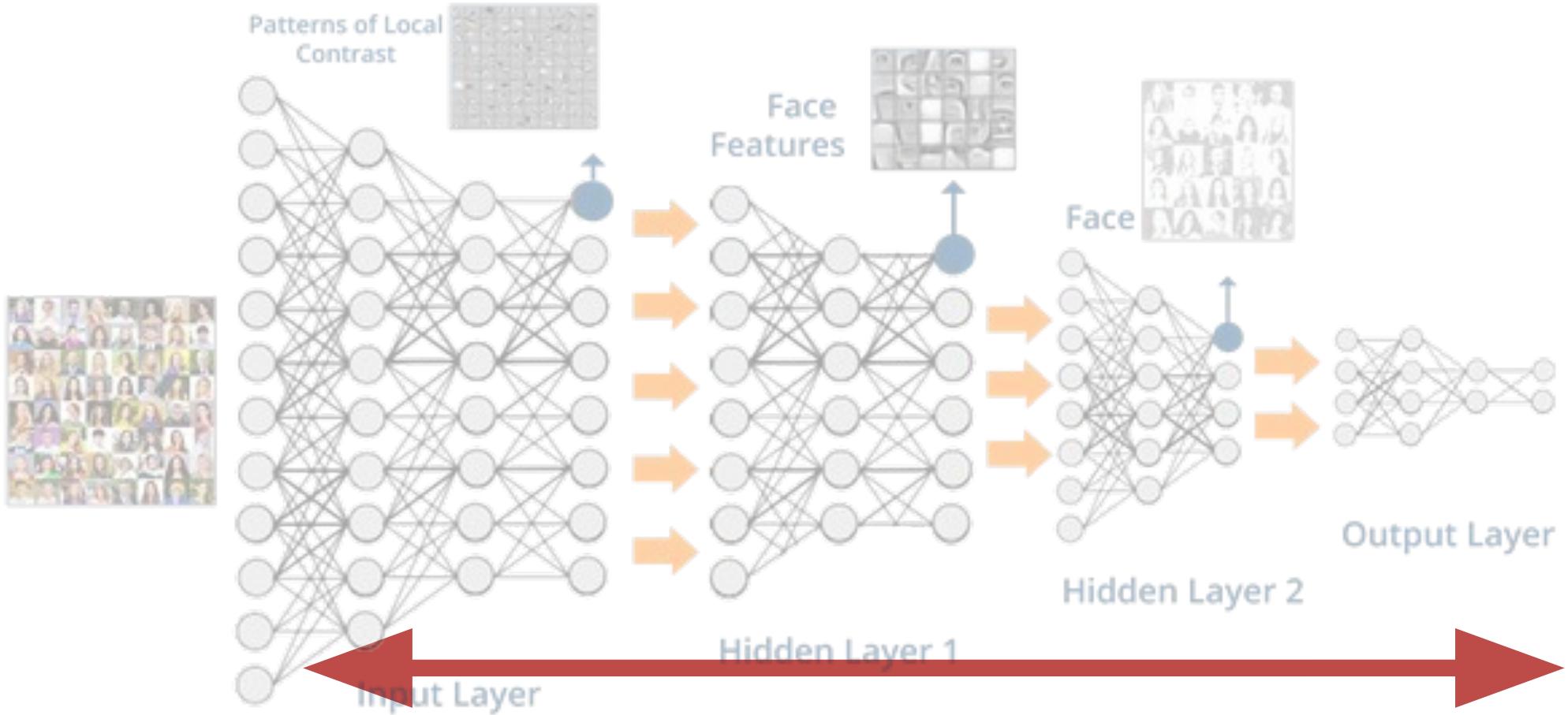
Deep L-Layers



Deep L-Layers

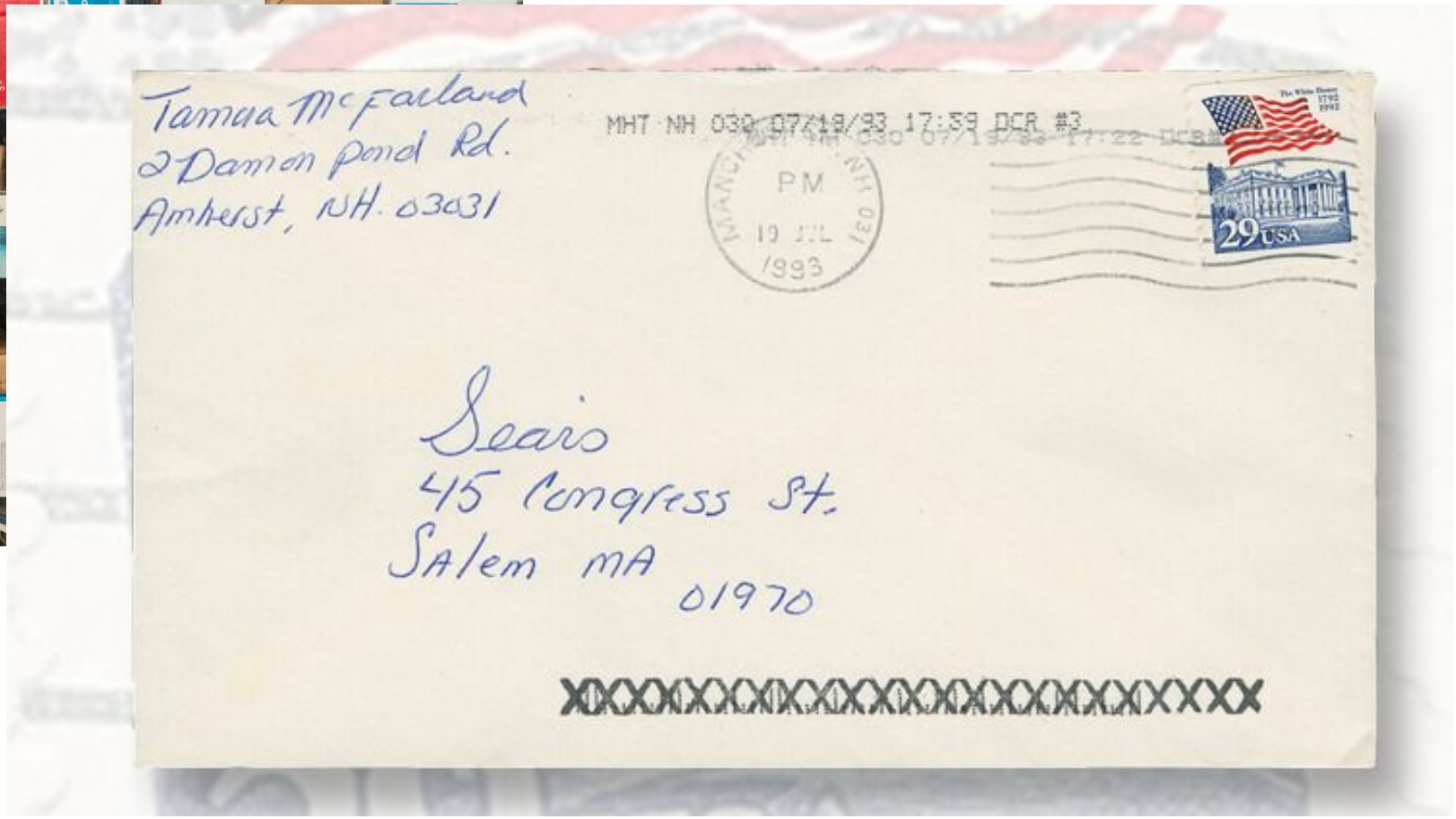


Deep L-Layers



Higher layers, higher levels of abstraction

AI – ML – Deep Learning – The US Postal Challenge



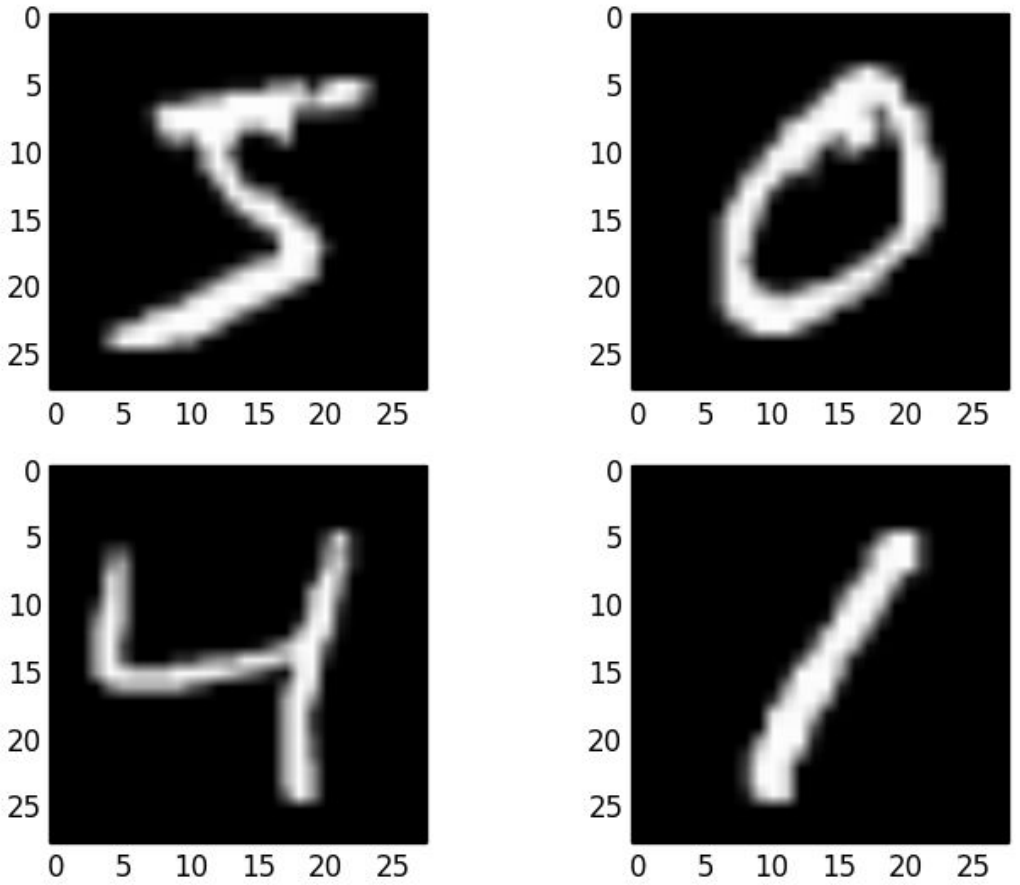
ANNs: The first challenge: MNIST

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples.

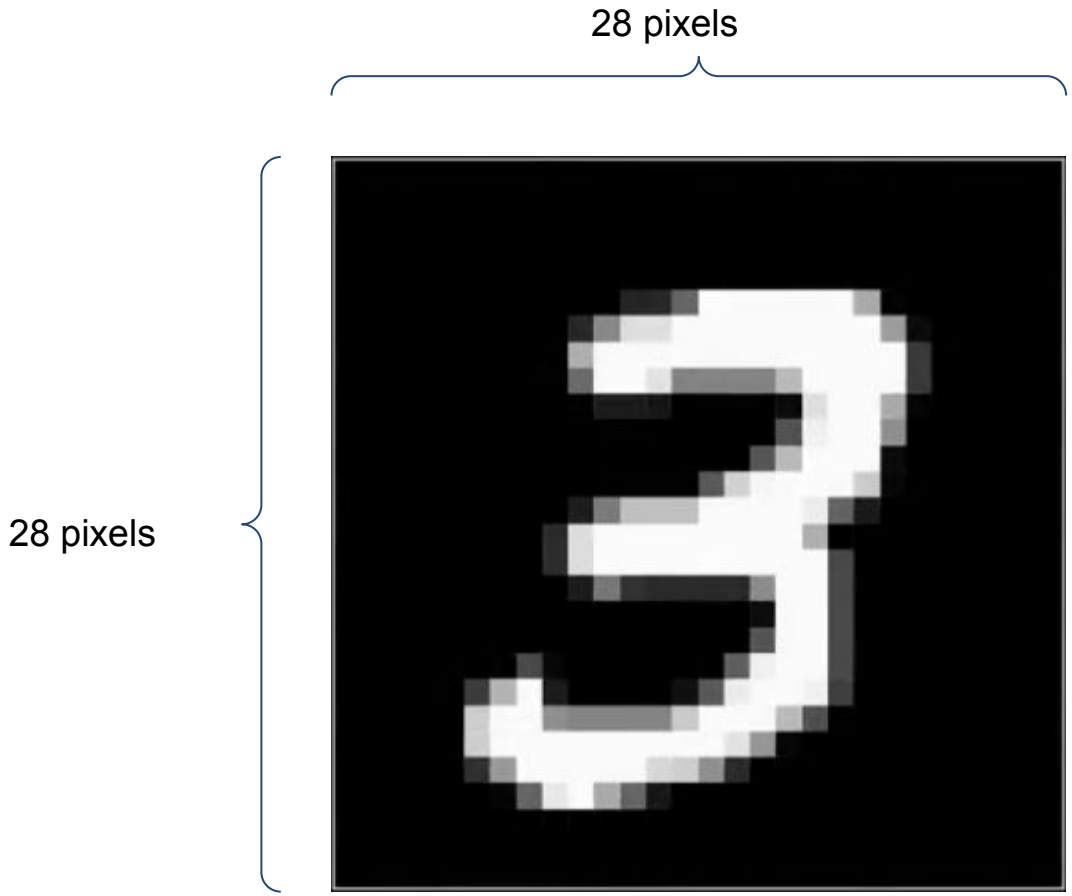
It is a very well-known benchmark for machine learning techniques

Each image is formed by 28x28 grayscale pixels

ANNs: The first challenge: MNIST



Deep L-Layers



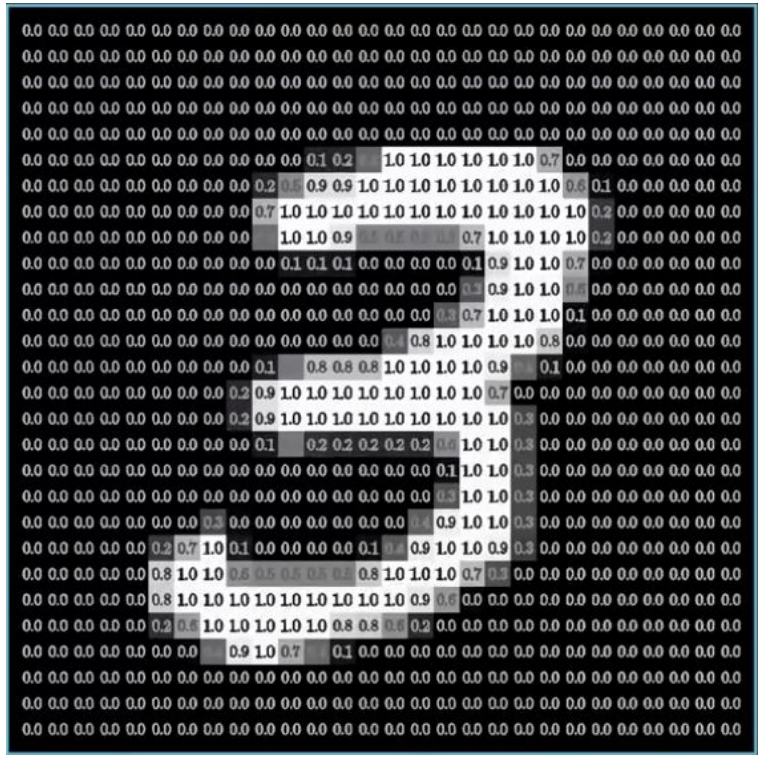
Let's get started with a real task:

- MNIST
- This is a 3… or a pixel array, depending on how you look at it
- Easy task for the brain but… How would you code a script to recognize this three?

Deep L-Layers

28 píxeles

28 píxeles



Let’s get started with a real task:

- MNIST
- This is a 3… or a pixel array, depending on how you look at it
- Easy task for the brain but… How would you code a script to recognize this three?

Deep L-Layers

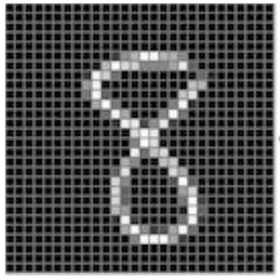
Deep L-Layers

- How can we tackle this problem with a DNN?

- Let us begin here:

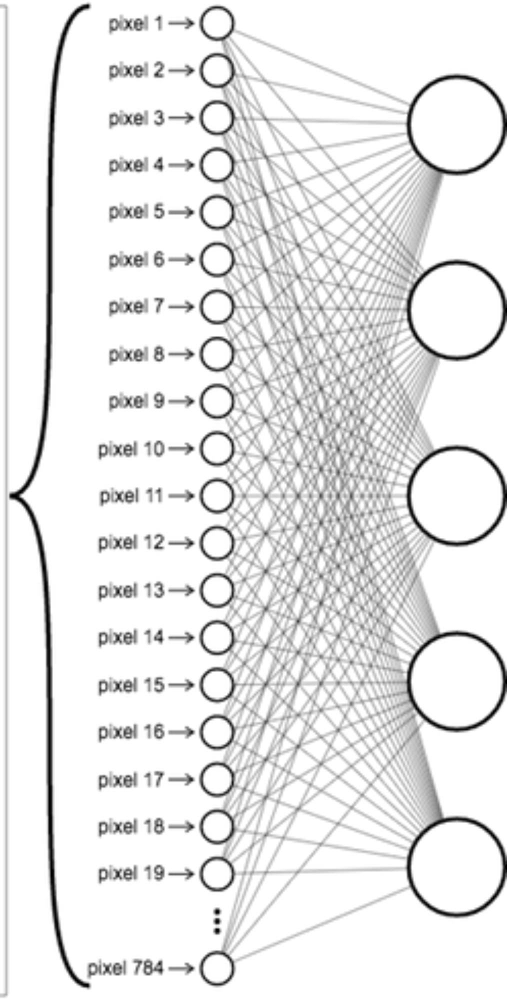
  - Standard DNN
  - Vanilla form, no rings nor bells
  - AKA Multilayer Perceptron

Artificial Neural Networks: Input MNIST



28 x 28
784 pixels

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Deep L-Layers

- Step by step… Neural Networks

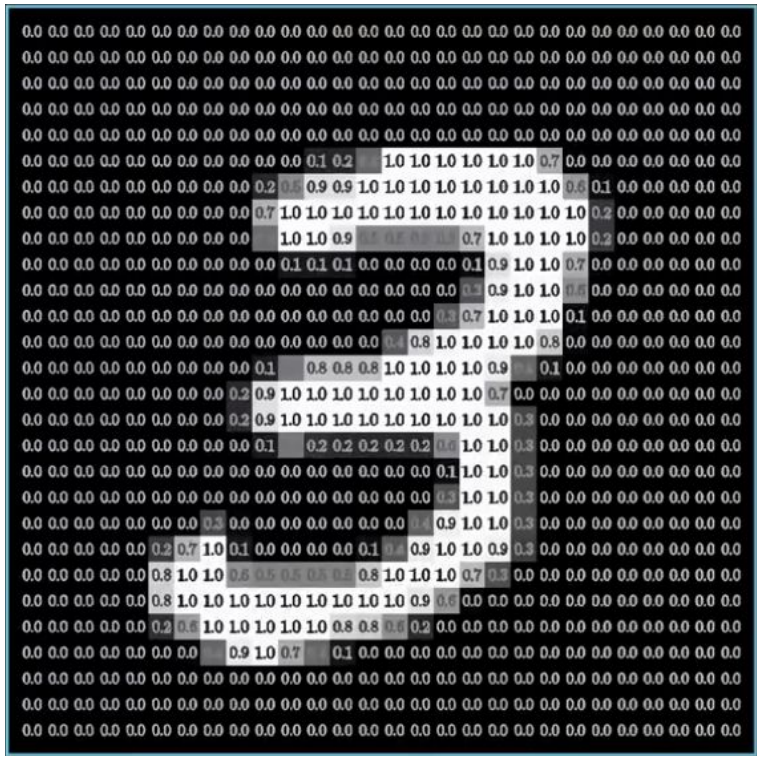
  - What is a neuron? So far, let's think of neurons as a container with some number.

  - How are they connected: Let's discover it!

Deep L-Layers

28 pixels

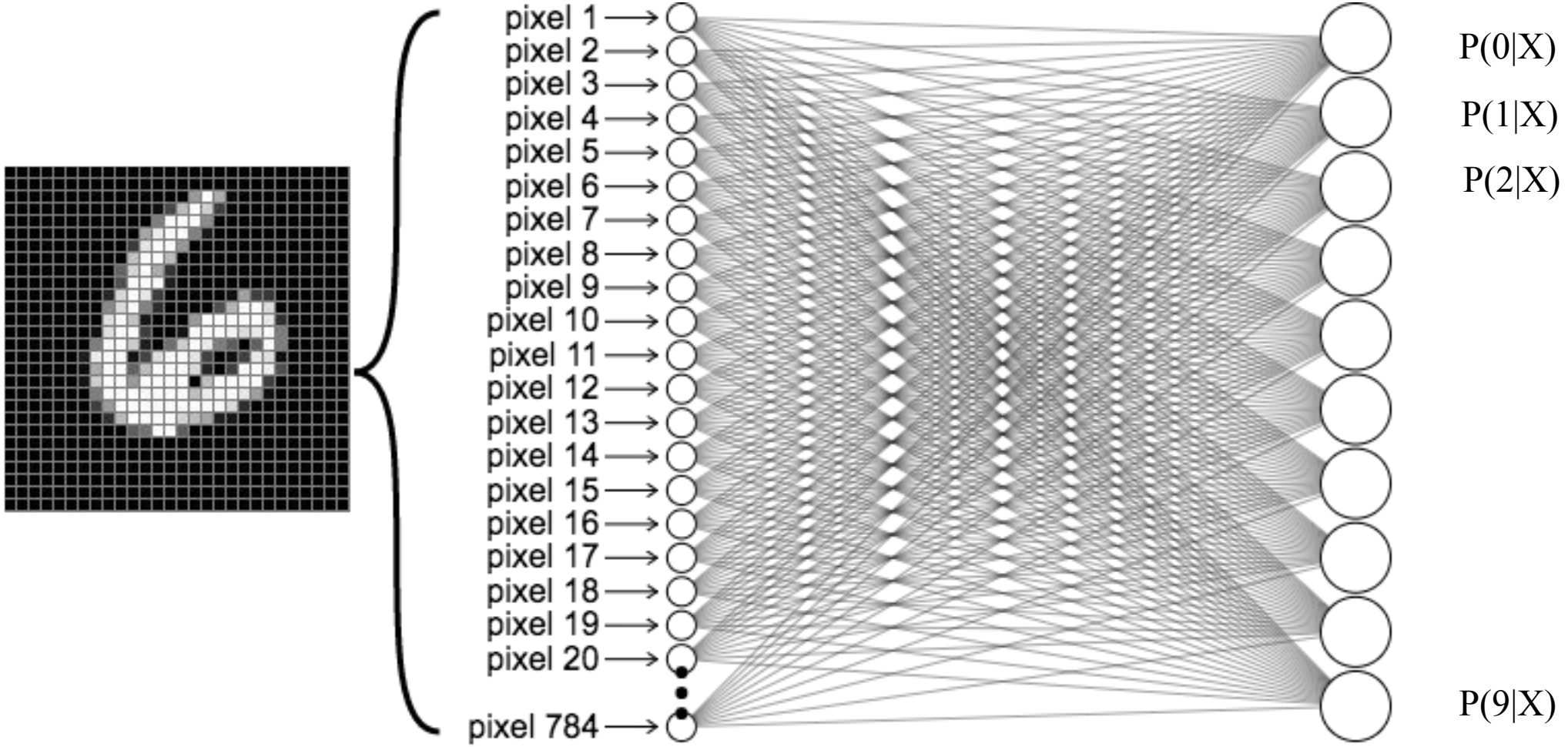
28 pixels



Input layer: 784 neurons

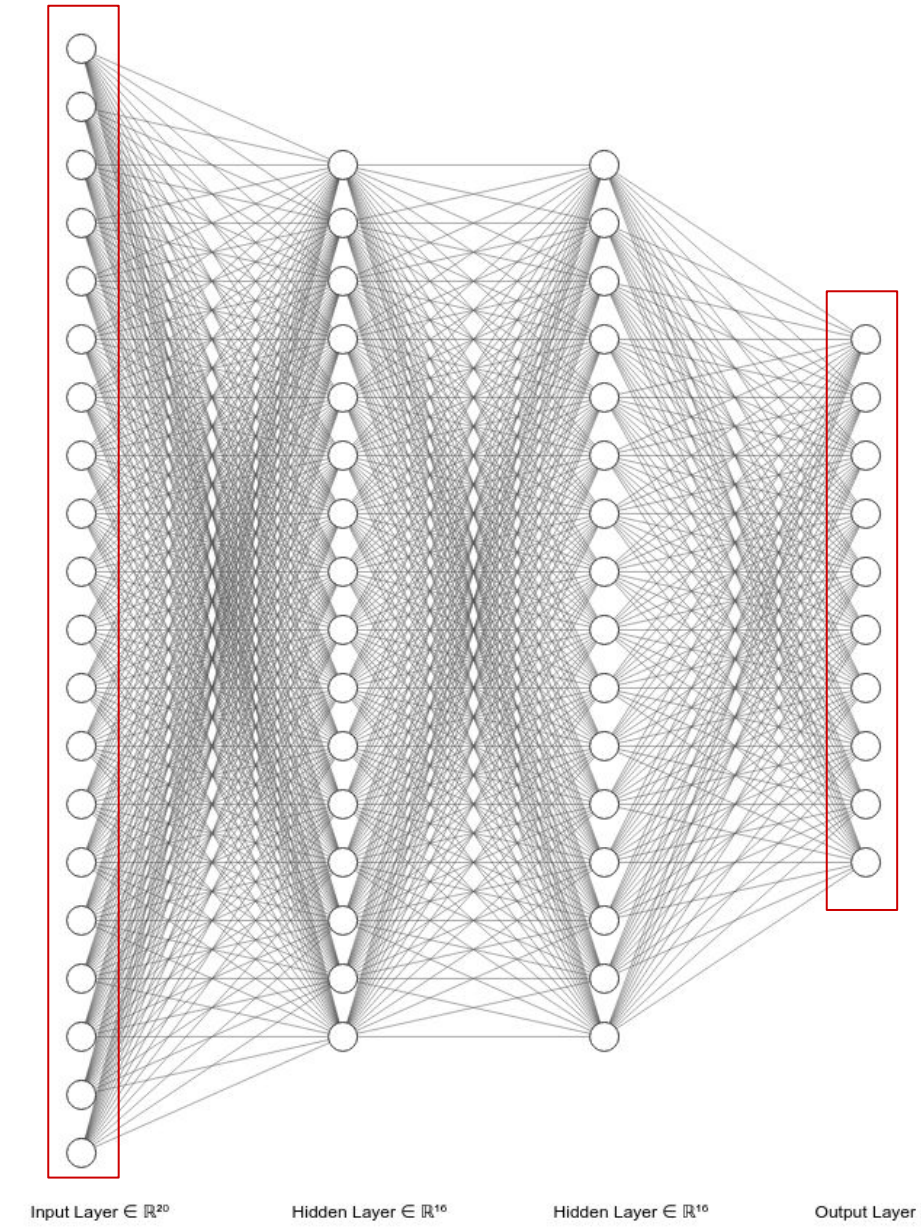
Output layer: 10 neurons.

Artificial Neural Networks: Output MNIST



Deep L-Layers

784 píxeles

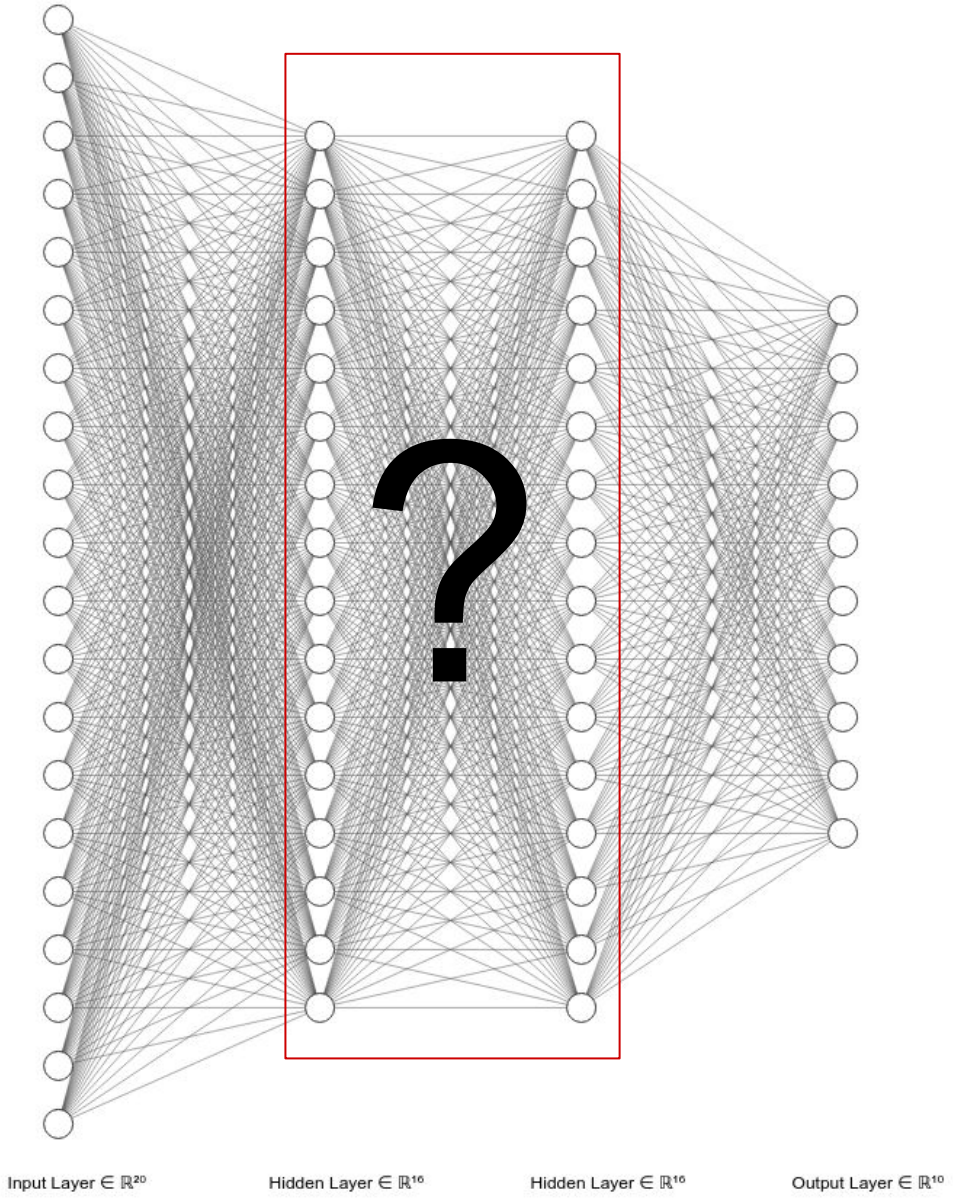


Input layer: 784 neurons

Output layer: 10 neurons.

Deep L-Layers

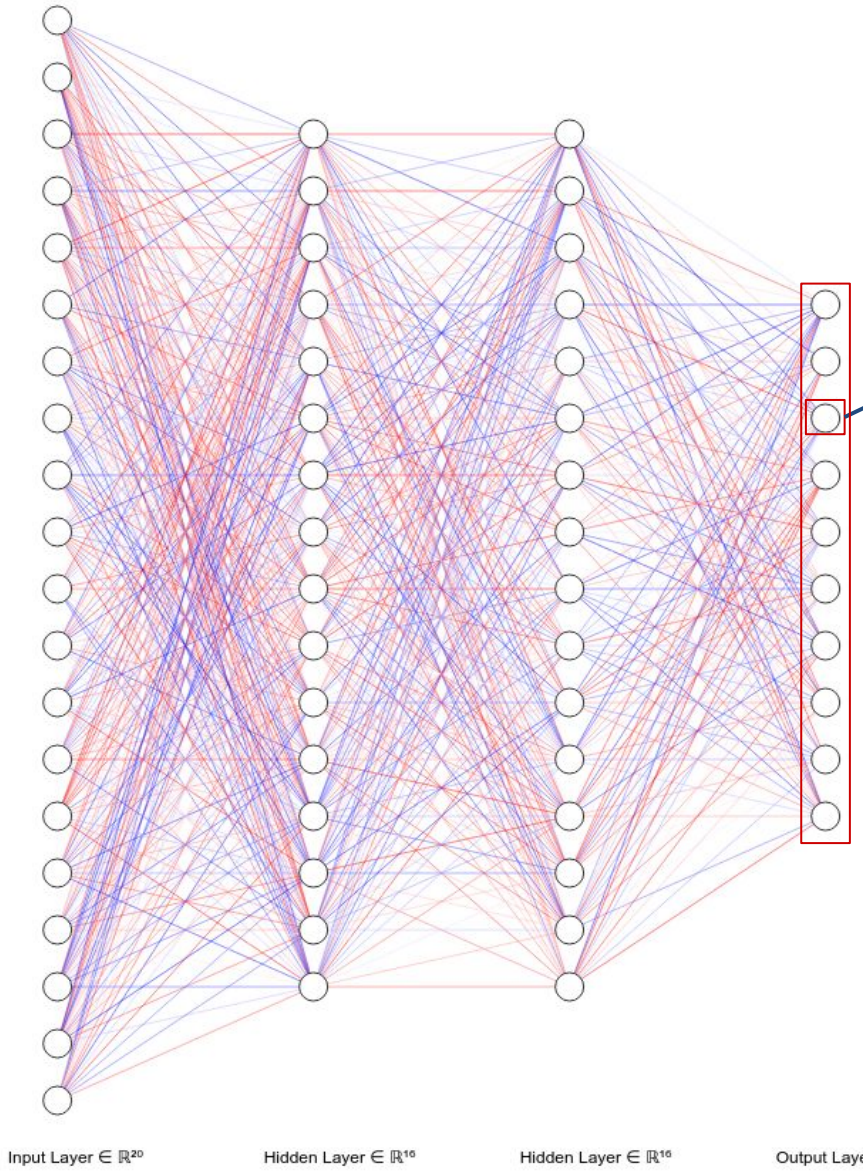
784 píxeles



Hidden Layers: ¿?

Deep L-Layers

784 pixels



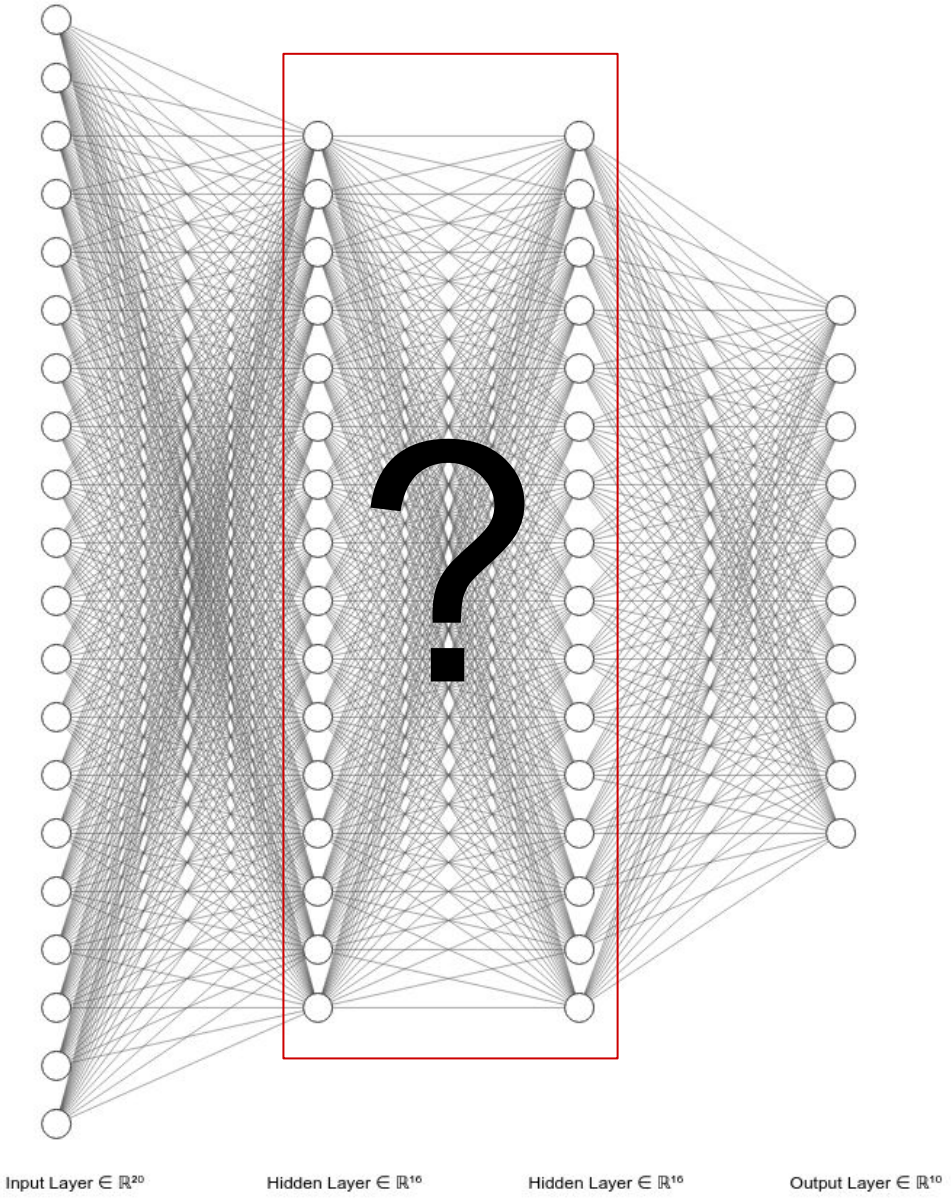
0.89

Network already trained.

Output layer tells us which number we have in the input

Deep L-Layers

784 píxeles



Remember:

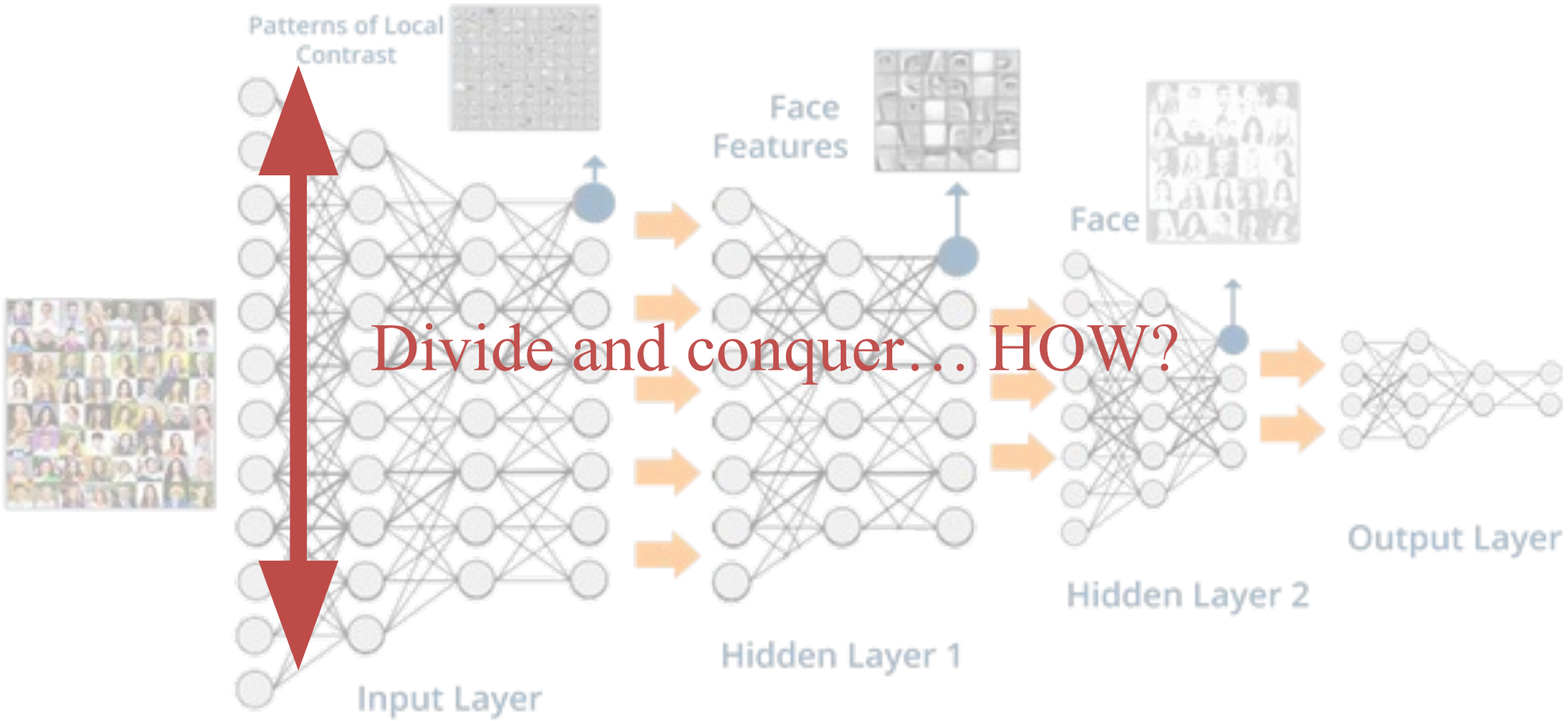
¿Why do we use layers?

- XOR
- Circuit theory

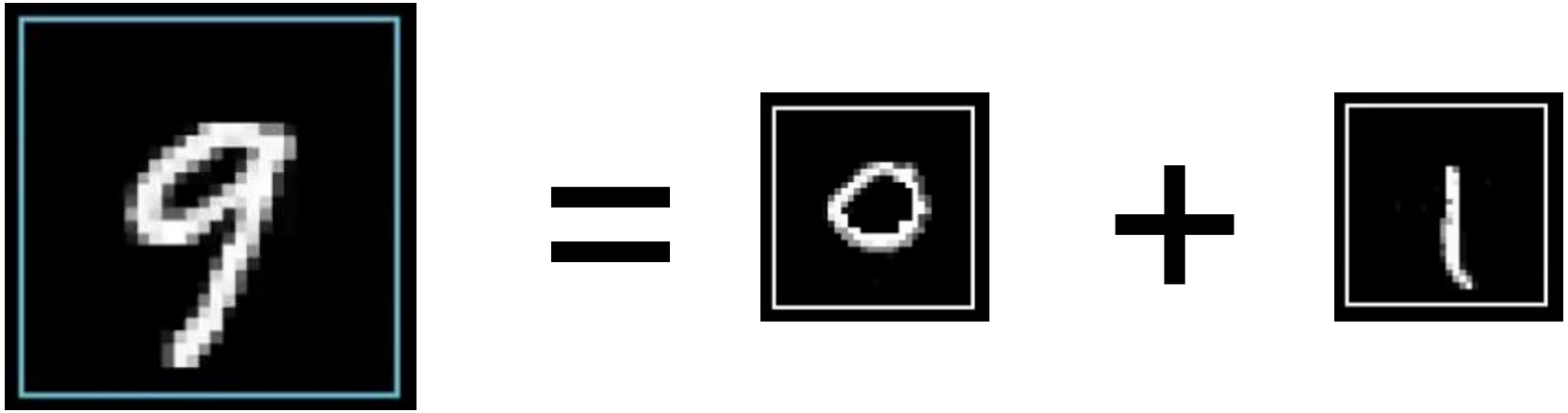
Outline

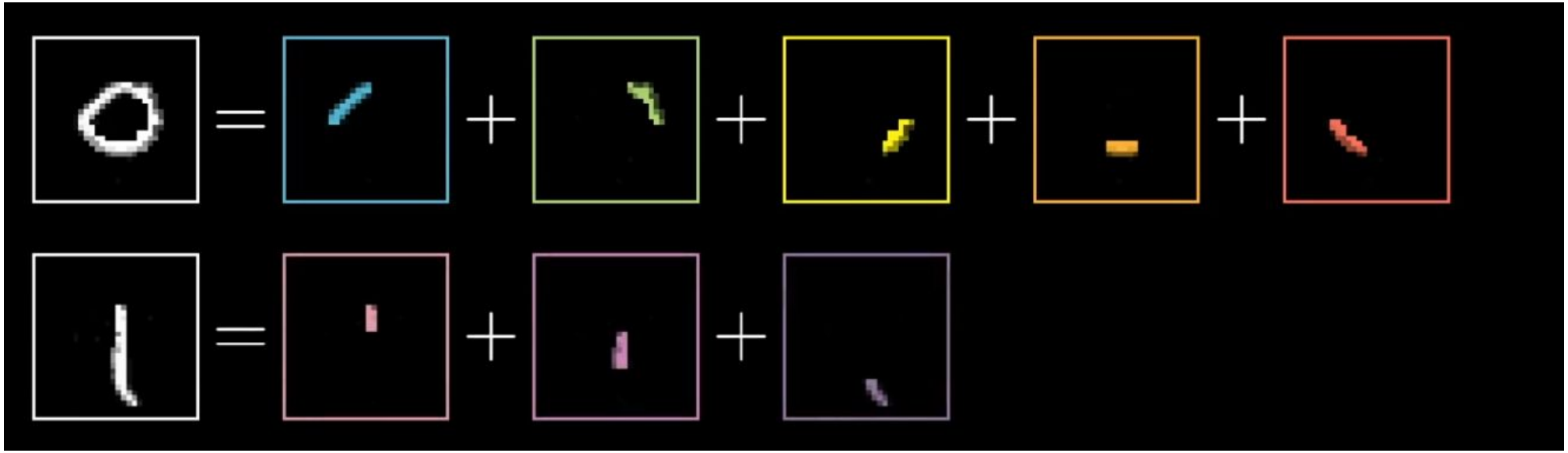
1. Recap
2. Deep L-Layers
3. Divide and Conquer
4. The Neuron: Pattern Recognition
5. DNNs

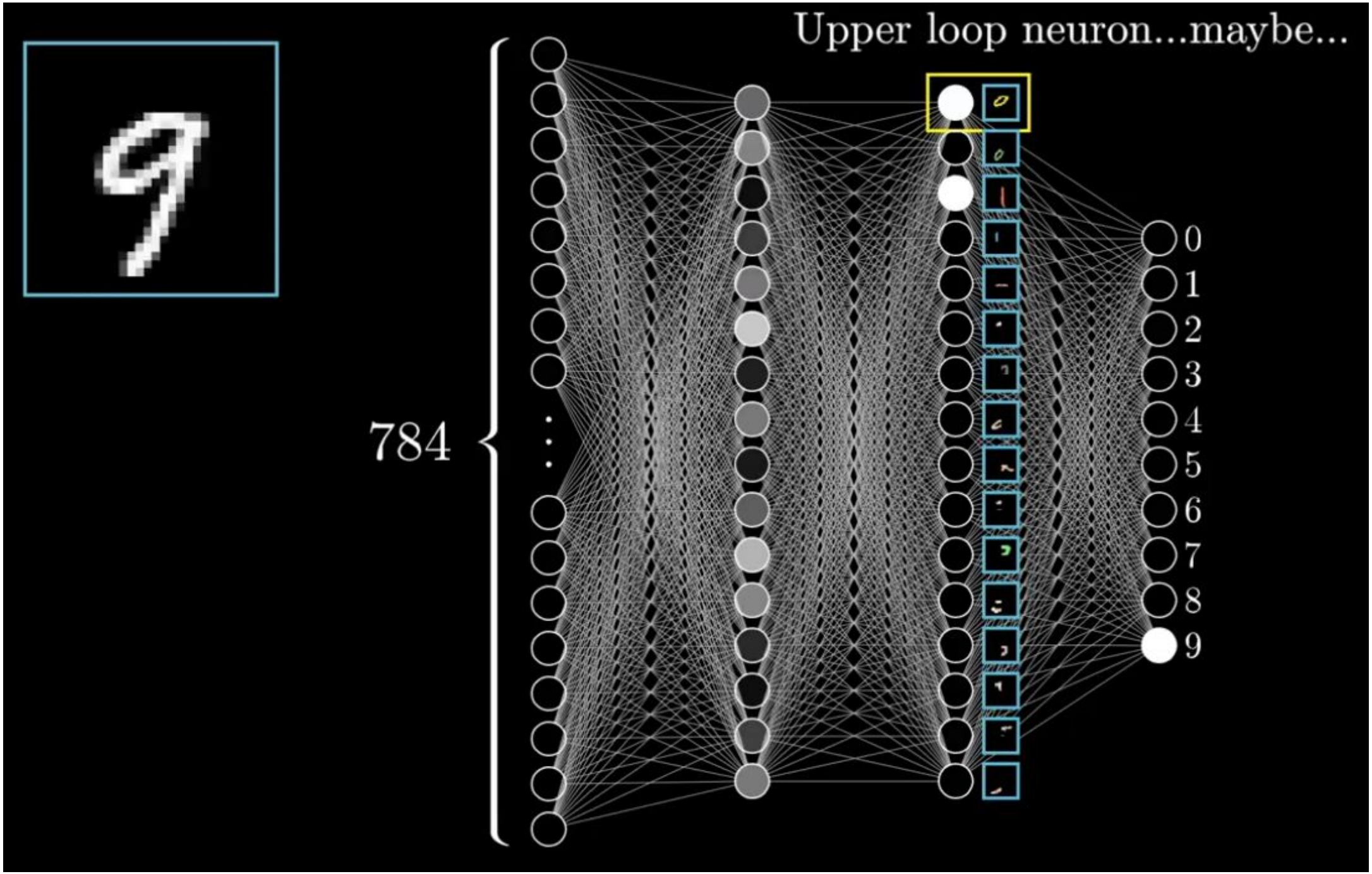
Divide and Conquer

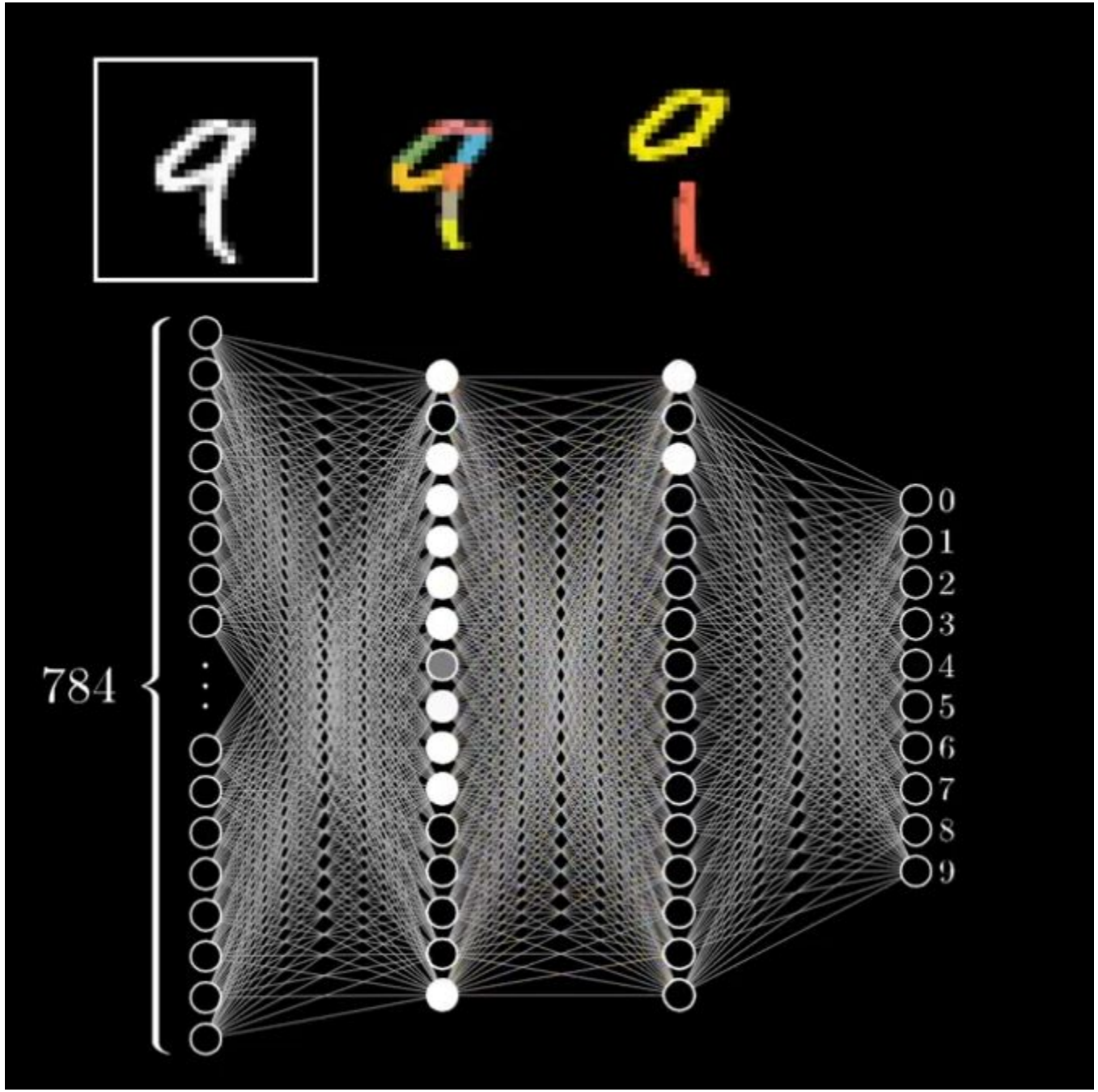


Divide and Conquer






Upper loop neuron...maybe...
784
0
1
2
3
4
5
6
7
8
9

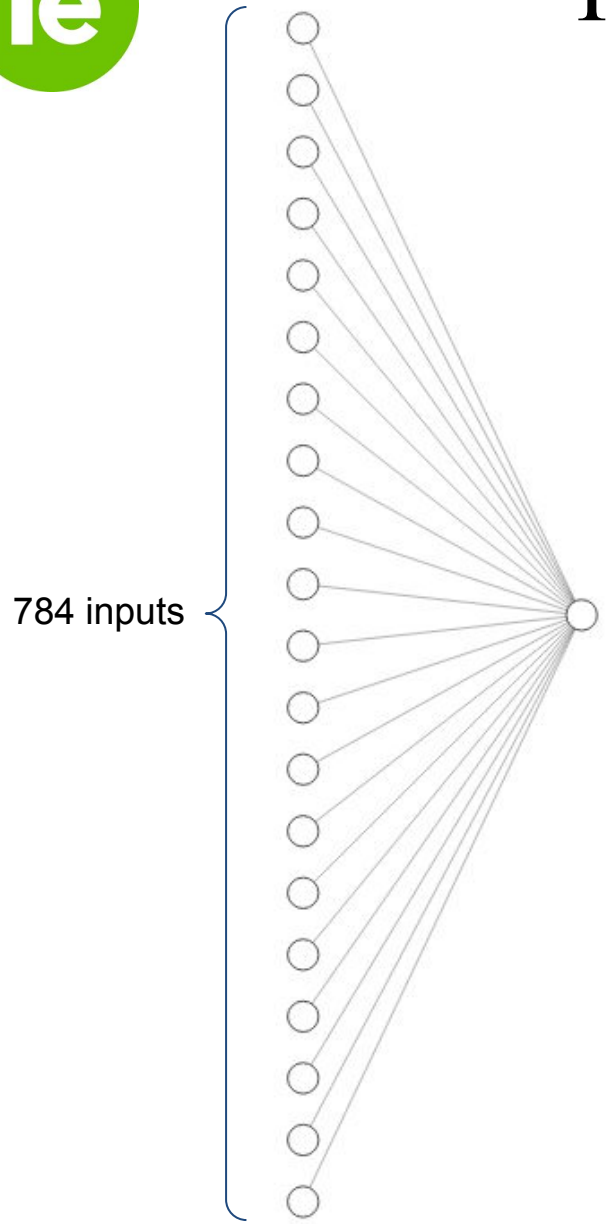

784
0
1
2
3
4
5
6
7
8
9

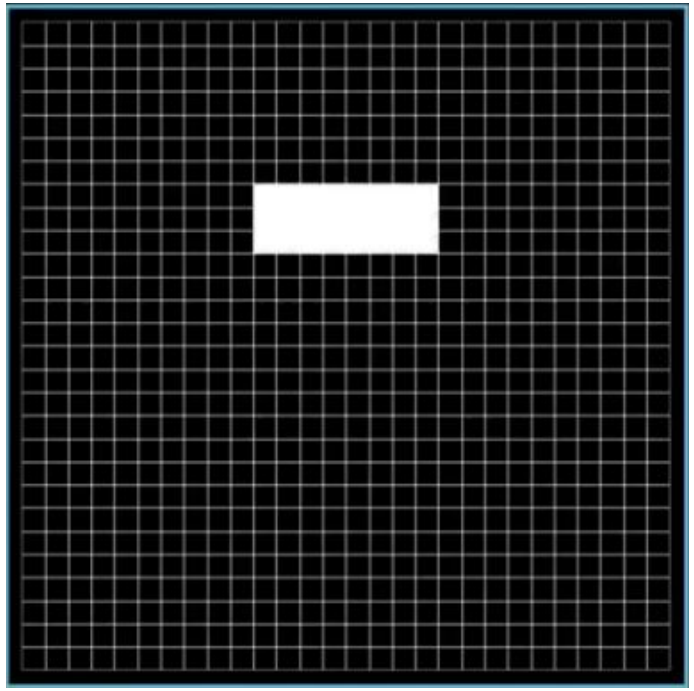
Divide and Conquer

Outline

1. Recap
2. Deep L-Layers
3. Divide and Conquer
4. The Neuron: Pattern Recognition
5. DNNs

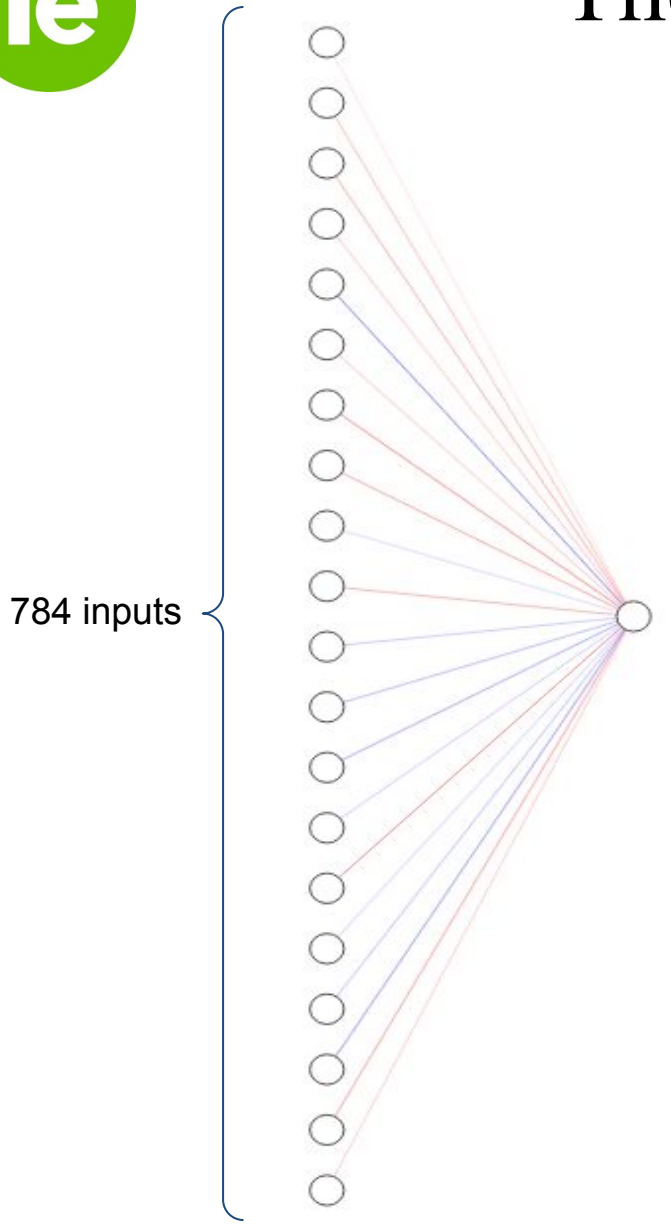
The Neuron: Pattern Recognition

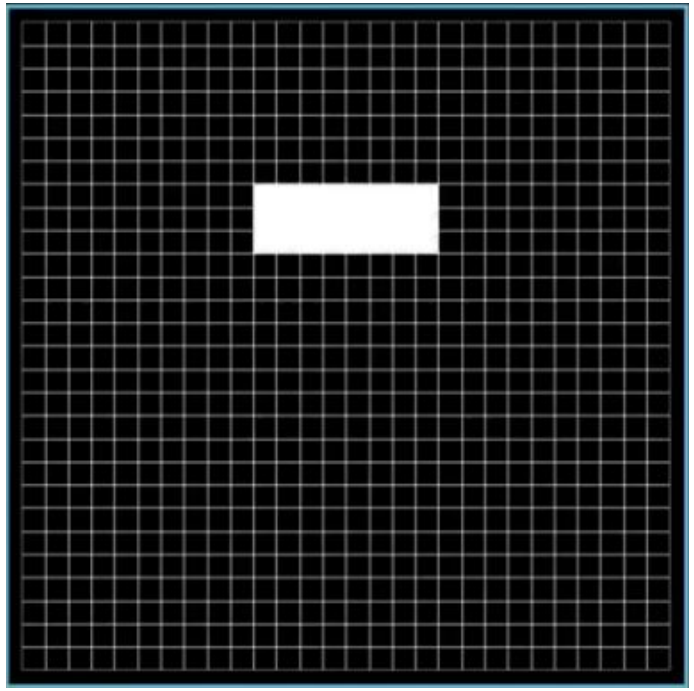




How can a neuron detect that pattern?

The Neuron: Pattern Recognition



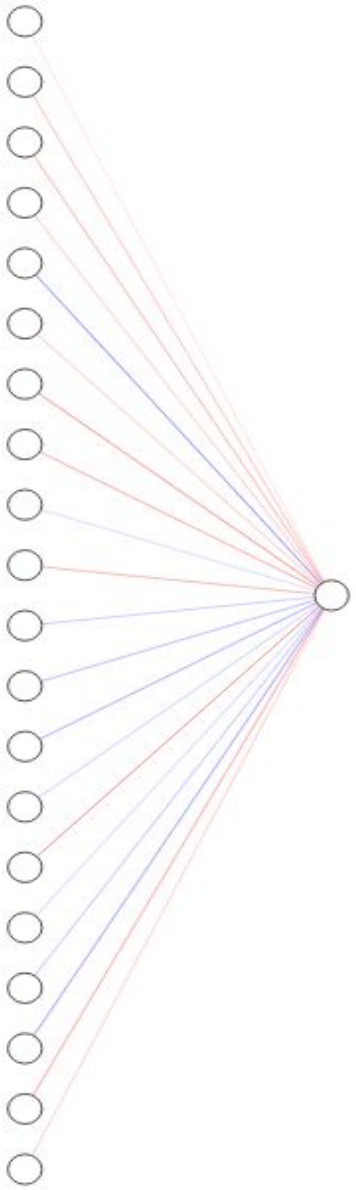


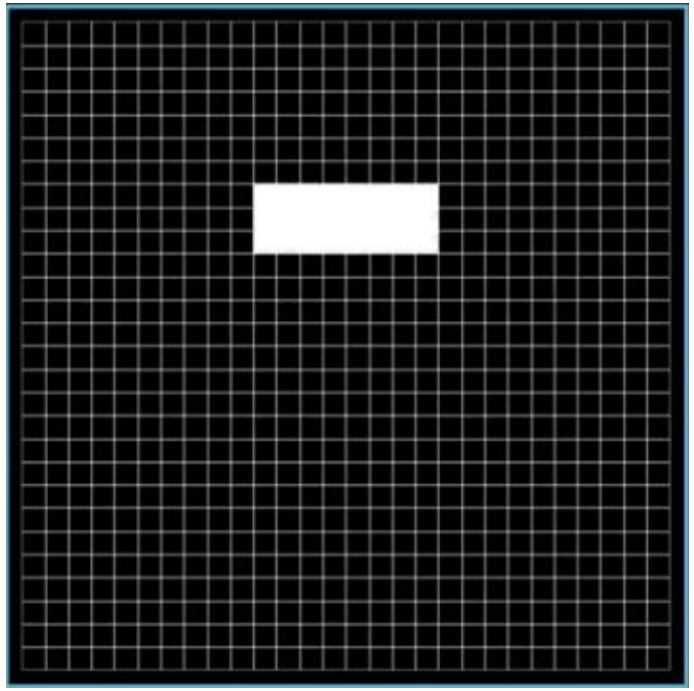
How can a neuron detect that pattern?

- Every pixel is an input
- What's the output?

The Neuron: Pattern Recognition

784 inputs





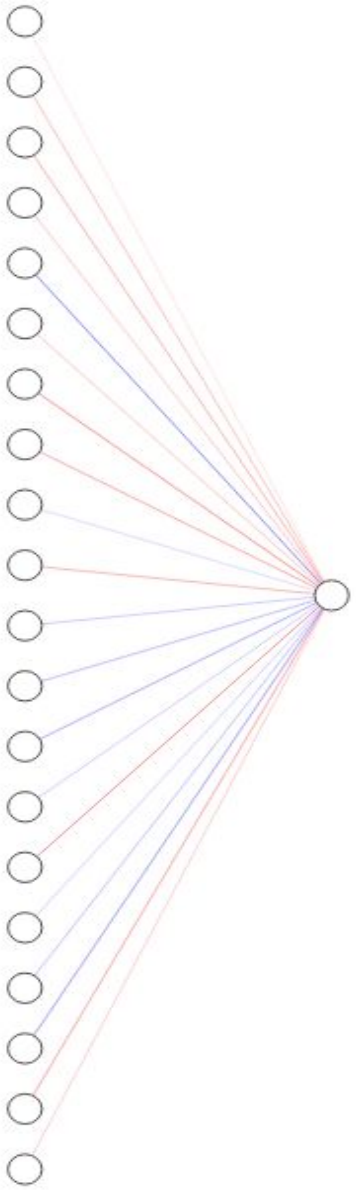
How can a neuron detect that pattern?

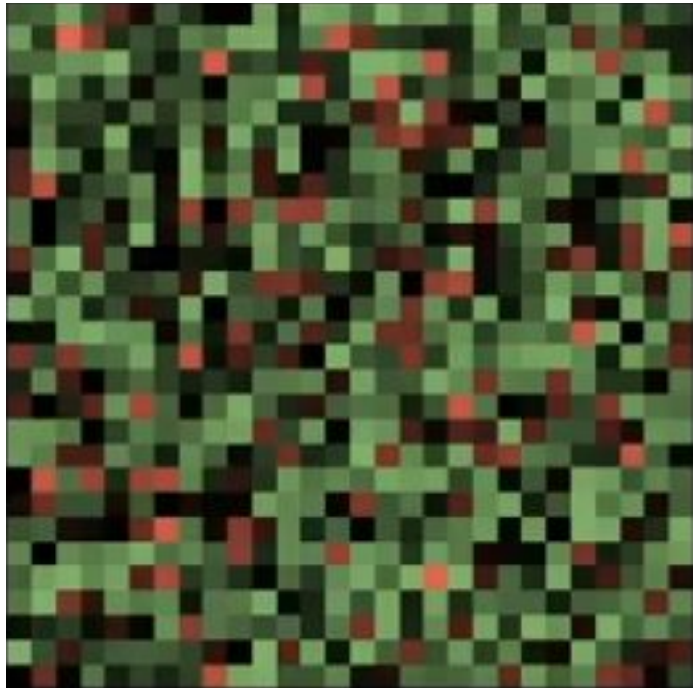
- Every pixel is an input
- What's the output?

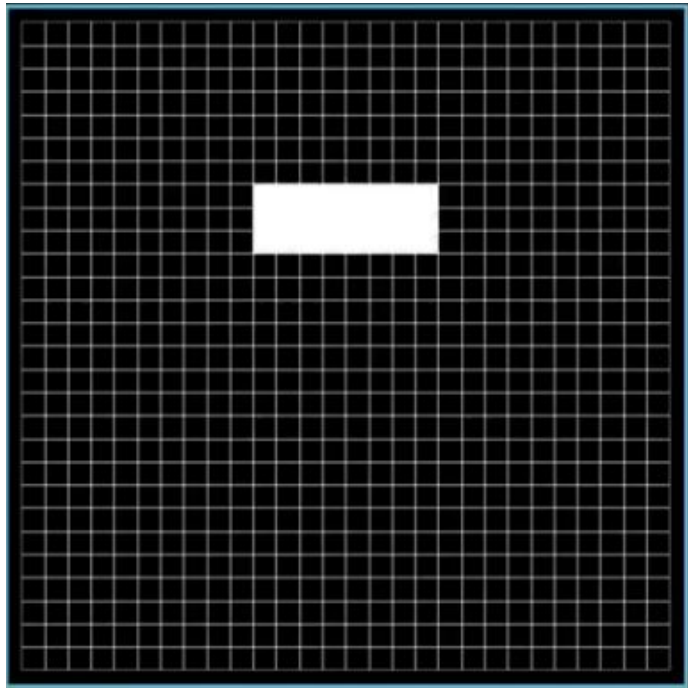
$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

The Neuron: Pattern Recognition

784 inputs



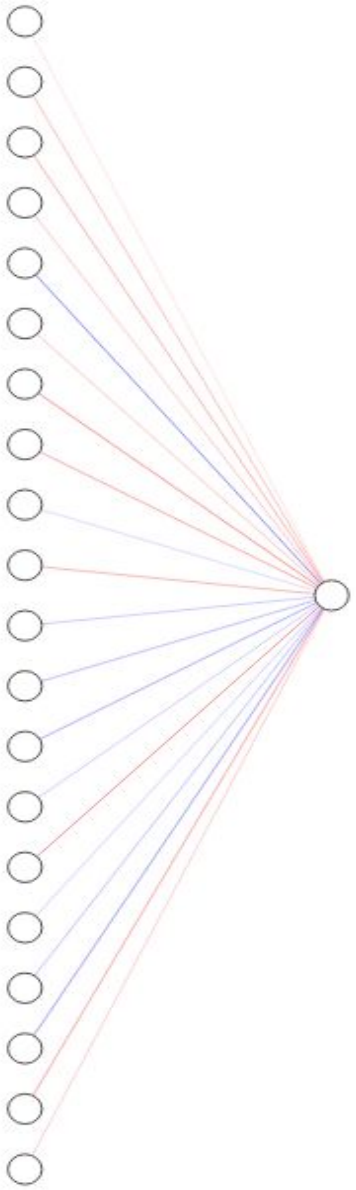




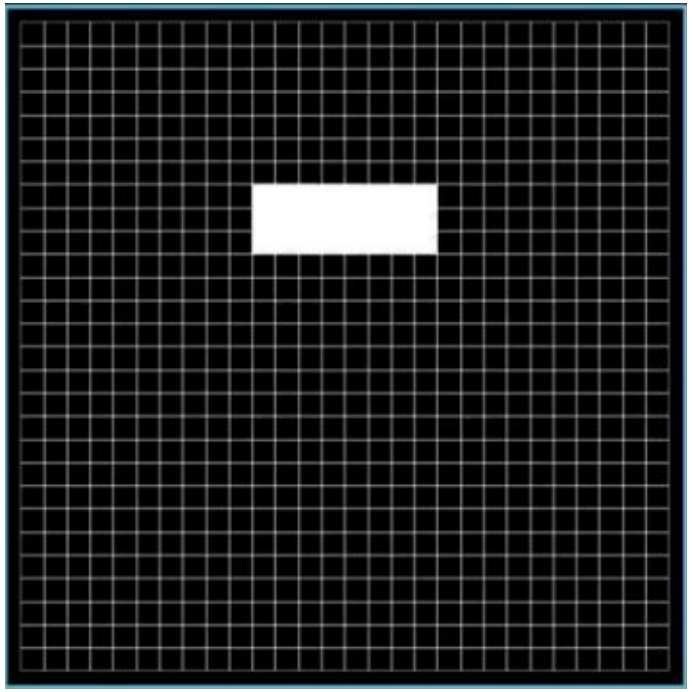
$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

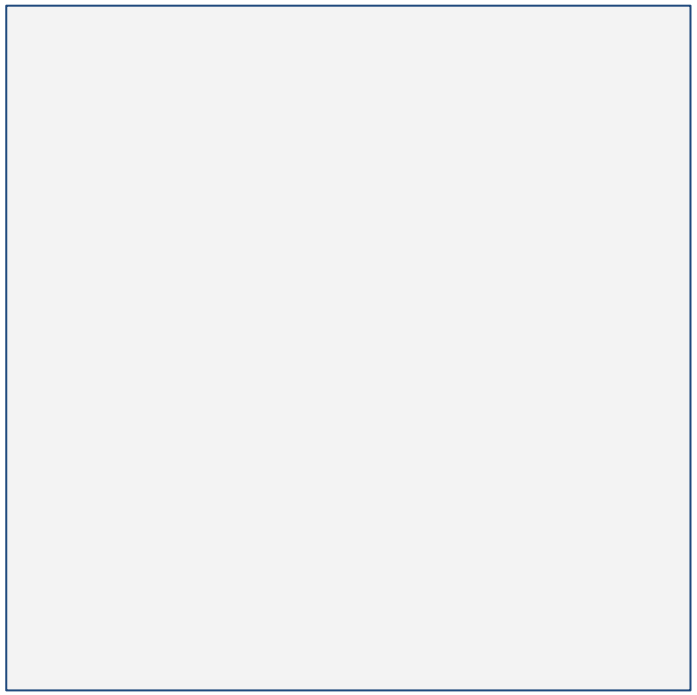
The Neuron: Pattern Recognition

784 inputs



- Watch out! There is a mistake here!

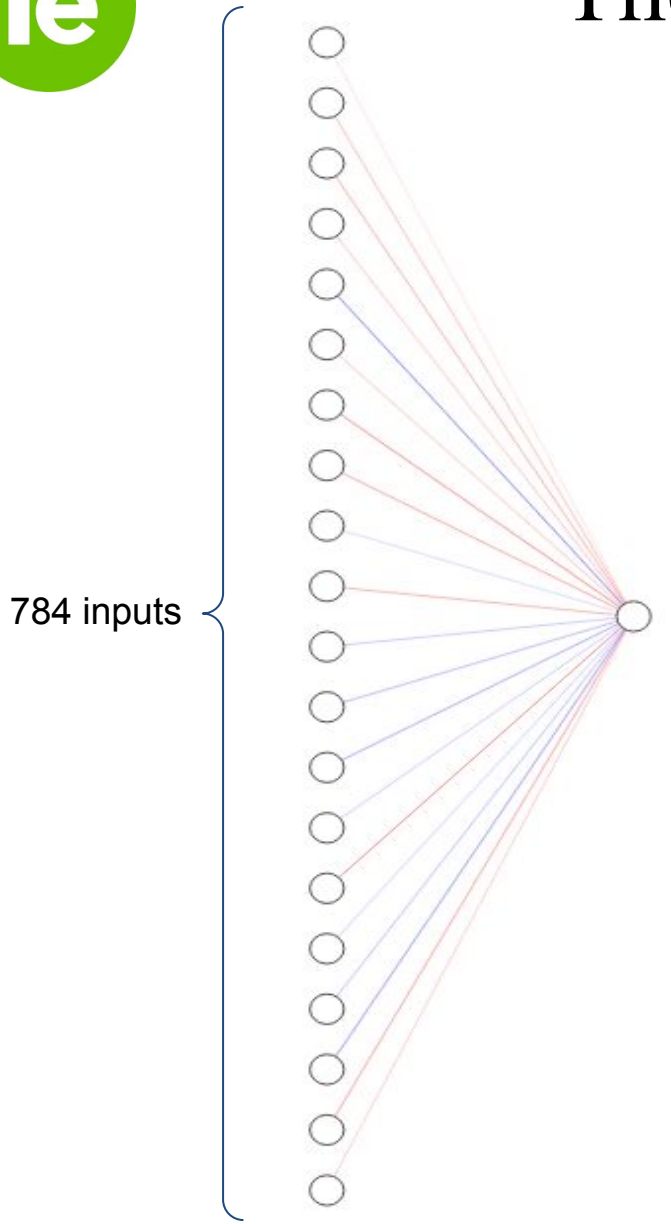




$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

The Neuron: Pattern Recognition

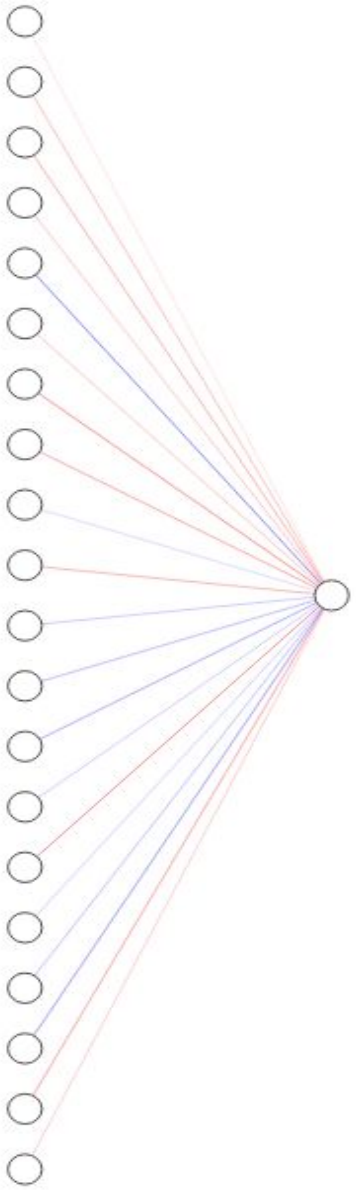
- Watch out! There is a mistake here!



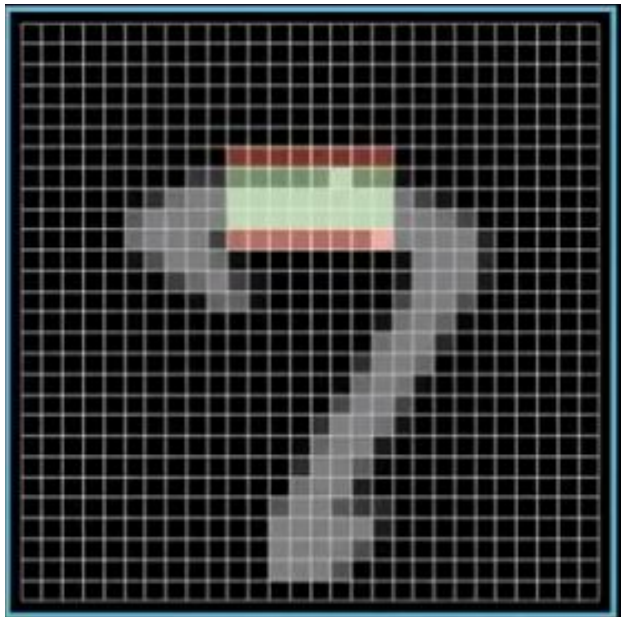
$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

The Neuron: Pattern Recognition

784 inputs



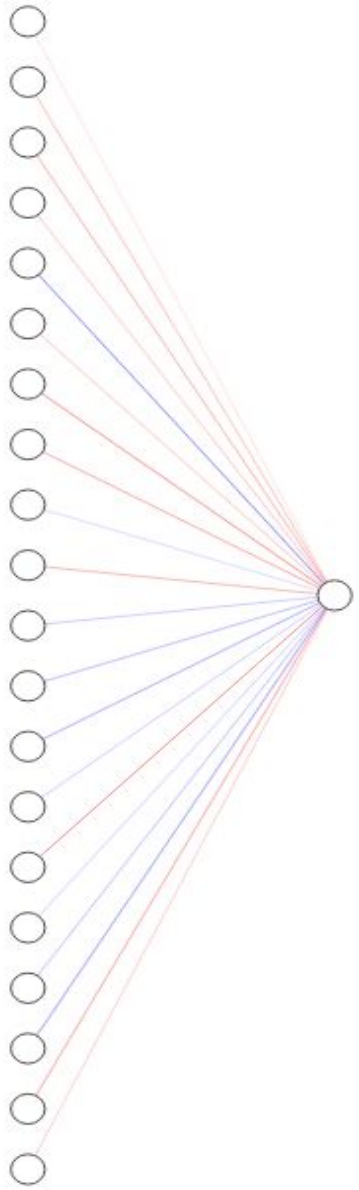
- Watch out! There is a mistake here!



$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

The Neuron: Pattern Recognition

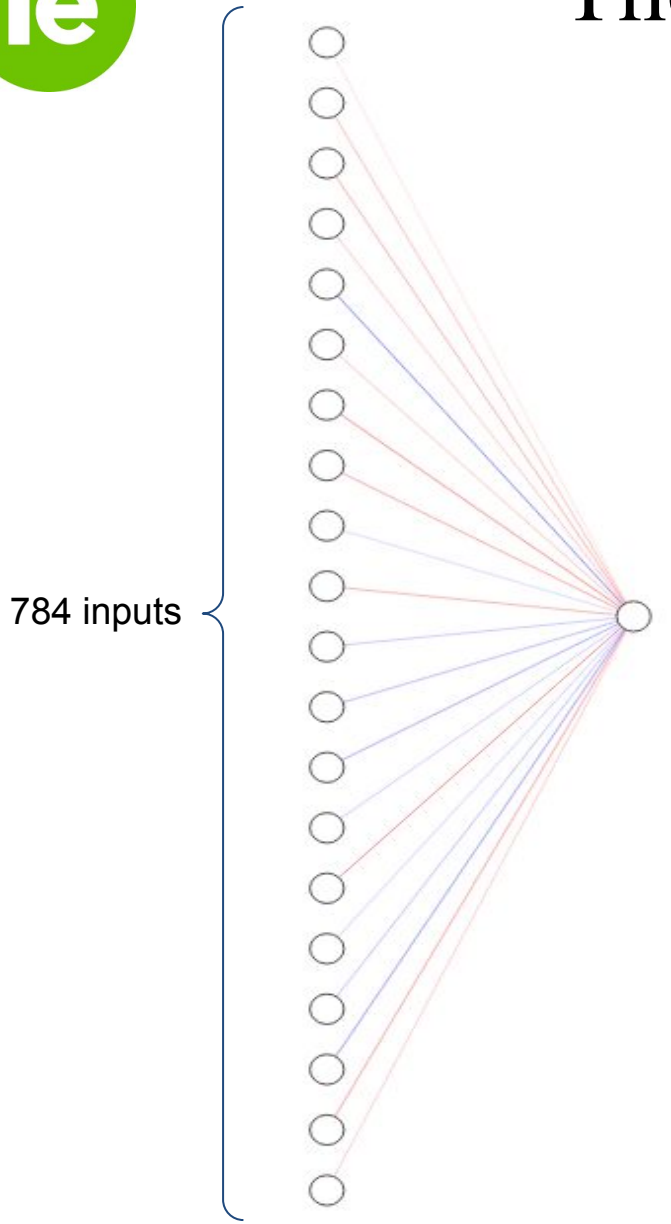
784 inputs



- What is our neuron output range?

$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

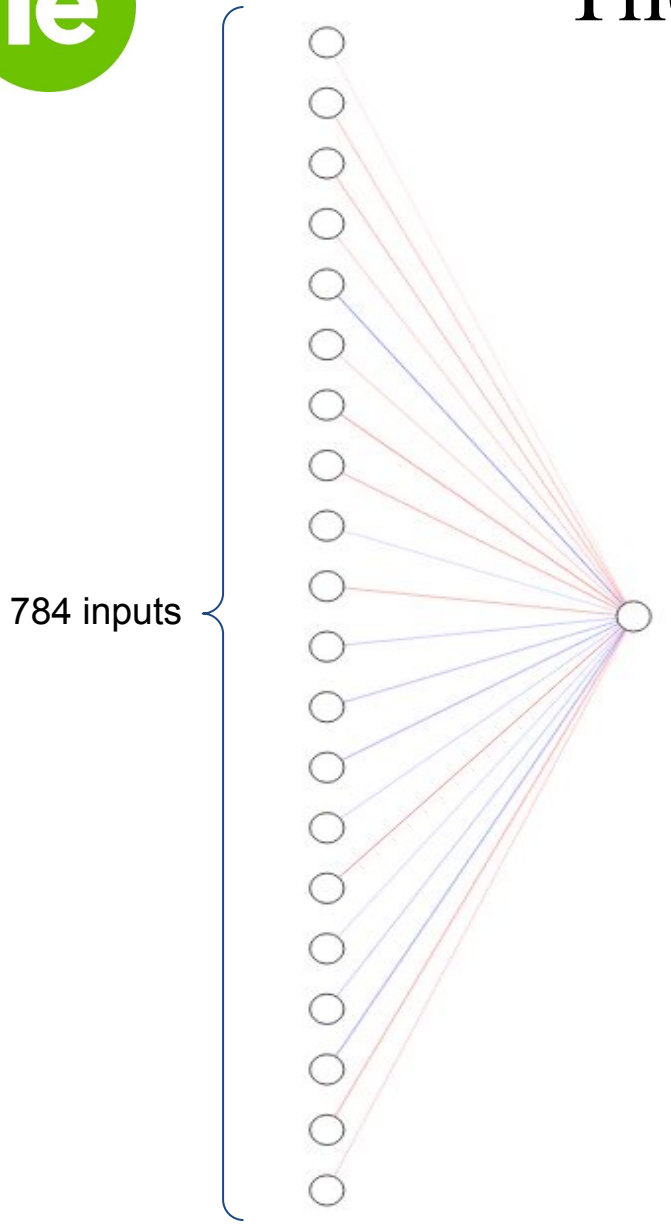
The Neuron: Pattern Recognition



- What is our neuron output range?
  - Real numbers

$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

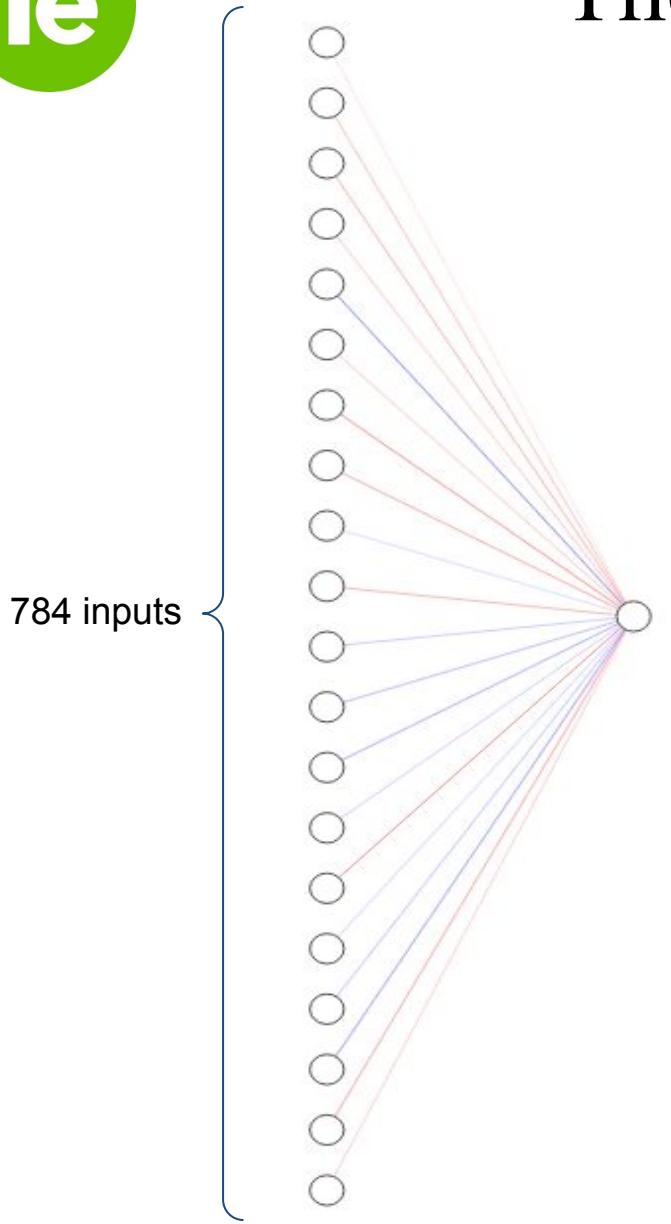
The Neuron: Pattern Recognition



- What is our neuron output range?
  - Real numbers
- What output range would we like?
  - Controlled range: (0,1) (-1,1).

$$a_n = i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} =$$

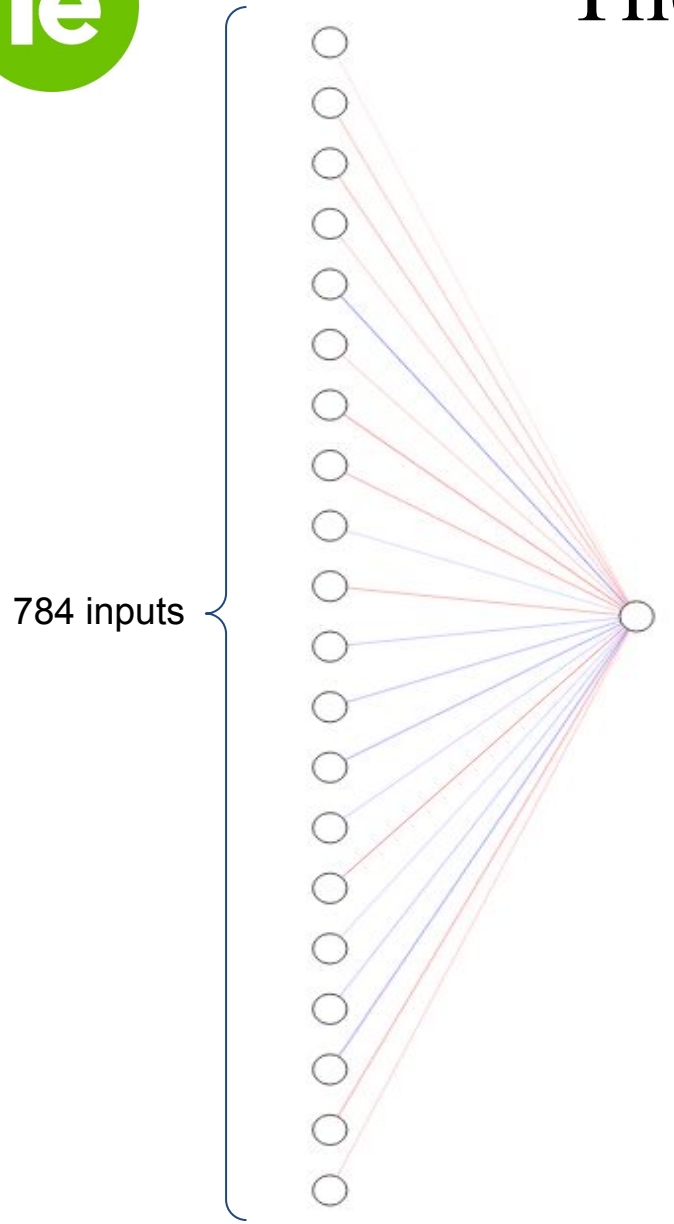
The Neuron: Pattern Recognition



- What is our neuron output range?
  - Real numbers
- What output range would we like?
  - Controlled range: (0,1) (-1,1).
  - Activation function (sigmoid, etc.)
- What if I want to be very sure that I have the exact pattern?

$$a_n = f(i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n}) =$$

The Neuron: Pattern Recognition



- What is our neuron output range?
  - Real numbers
- What output range would we like?
  - Controlled range: (0,1) (-1,1).
  - Activation function (sigmoid, etc.)
- What if I want to be very sure that I have the exact pattern?
  - Bias!

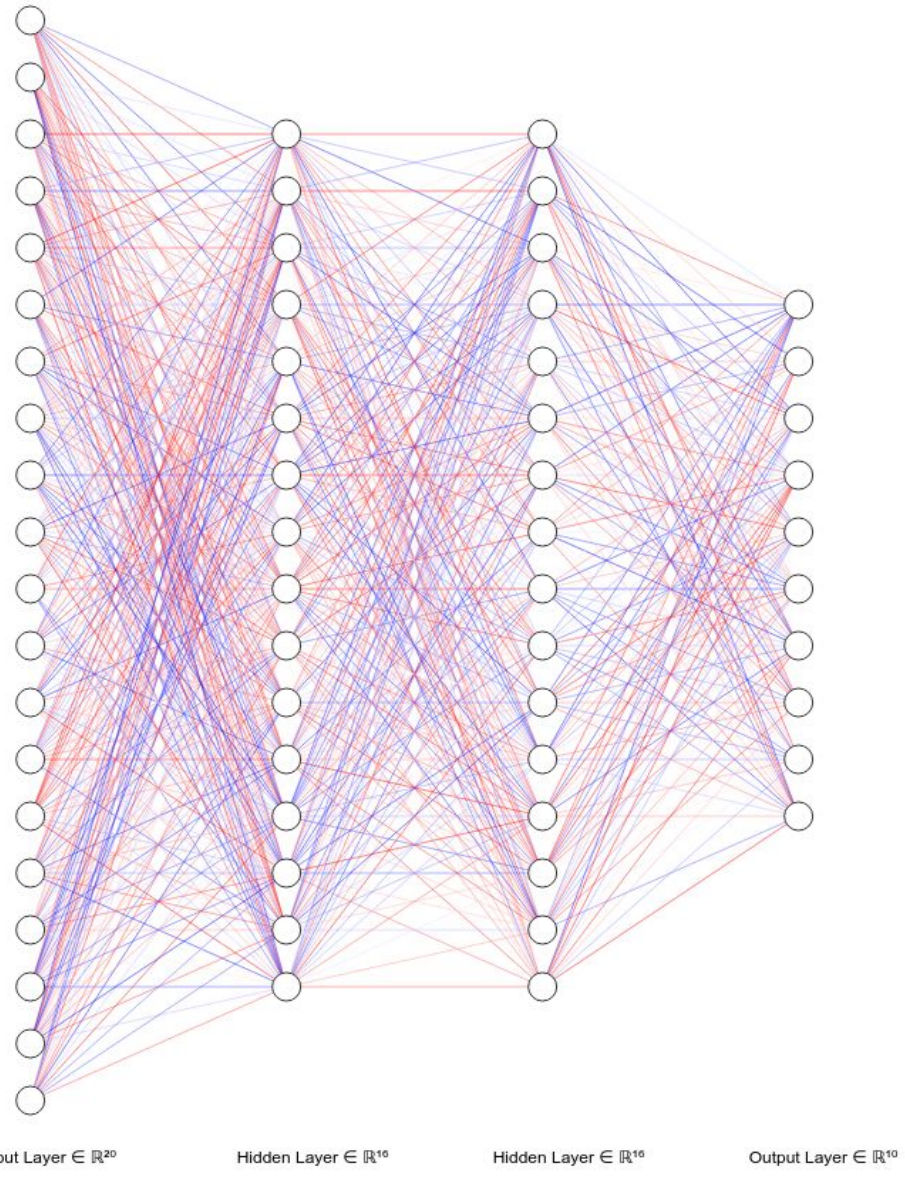
$$a_n = f(i_1 * w_{1n} + i_2 * w_{2n} + i_3 * w_{3n} + \ldots + i_{784} * w_{784n} + b_n)$$

Outline

1. Recap
2. Deep L-Layers
3. Divide and Conquer
4. The Neuron: Pattern Recognition
5. DNNs

DNNs

784 pixels



Complete Network:

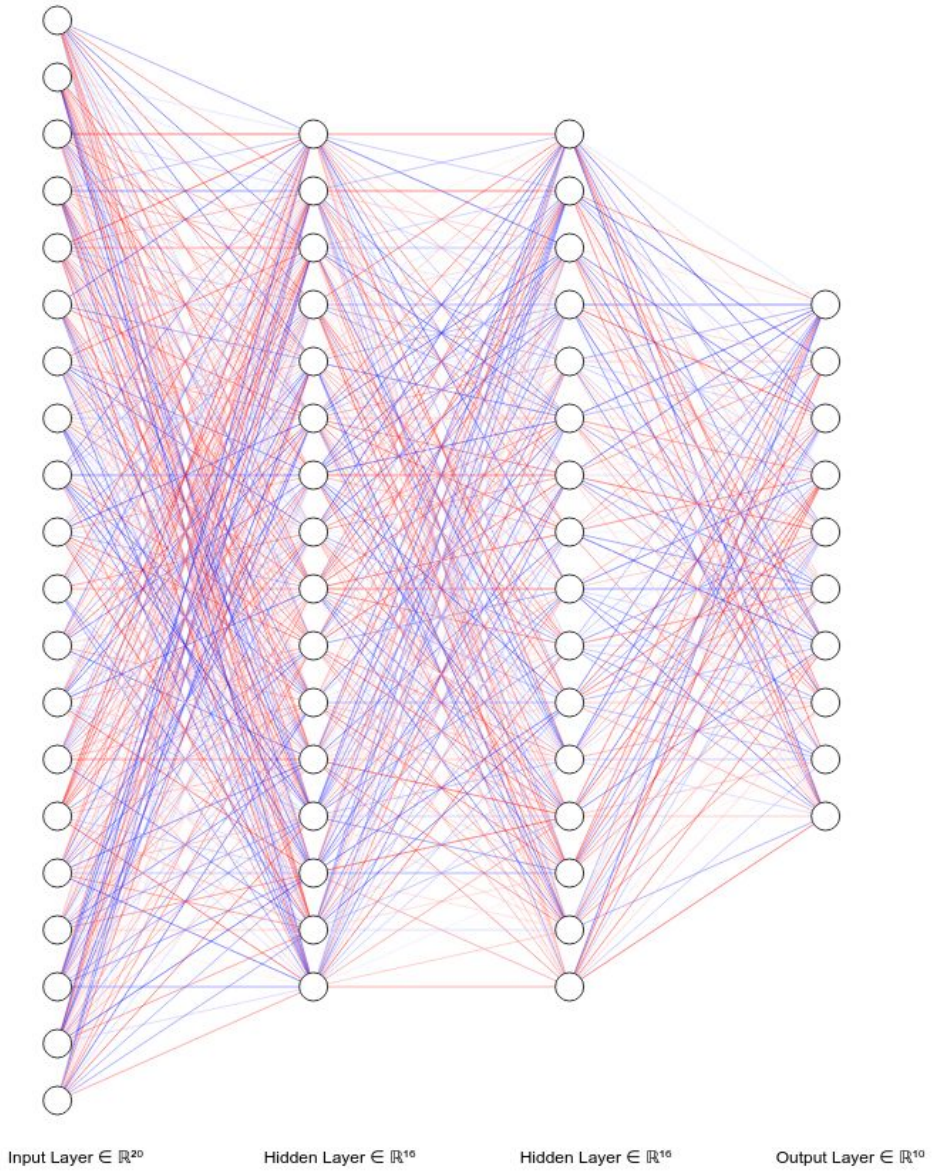
Weights:
784*16 + 16*16 + 16 * 10

Bias:
16+16+10

Trainable Params:
13002

DNNs

784 pixels

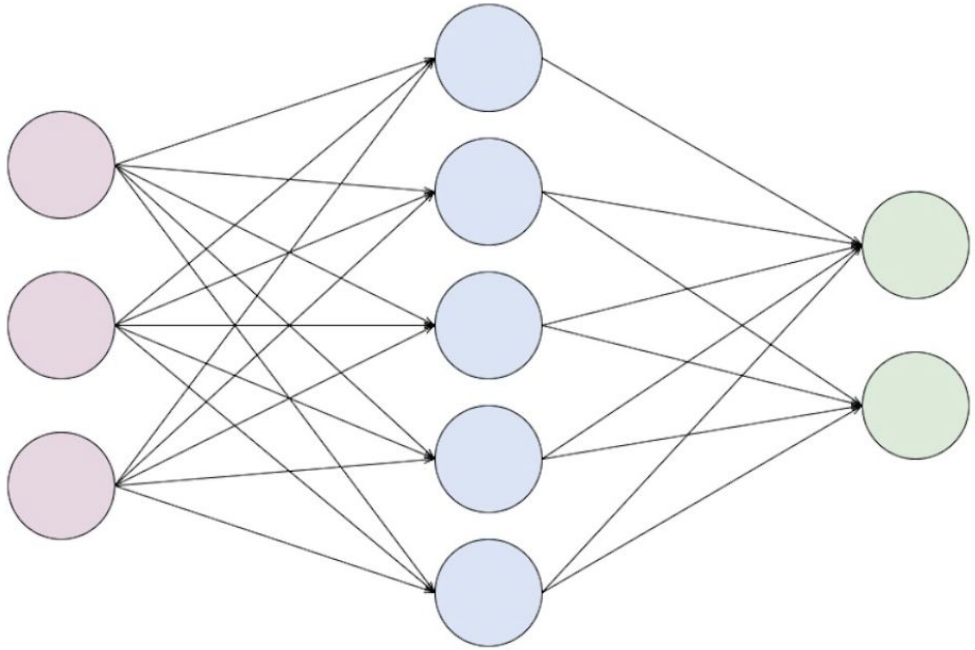


Complete Network:

Trainable Parameters:
13002

Training consist of finding the value for each trainable parameters that minimize the cost function on the training dataset

DNNs



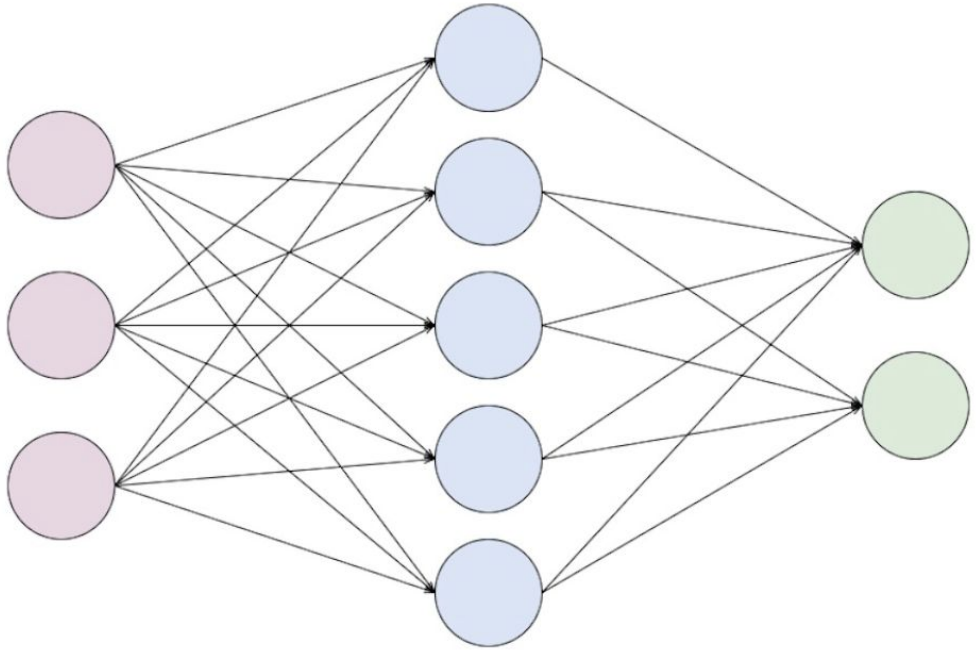
A net of information

DNNs

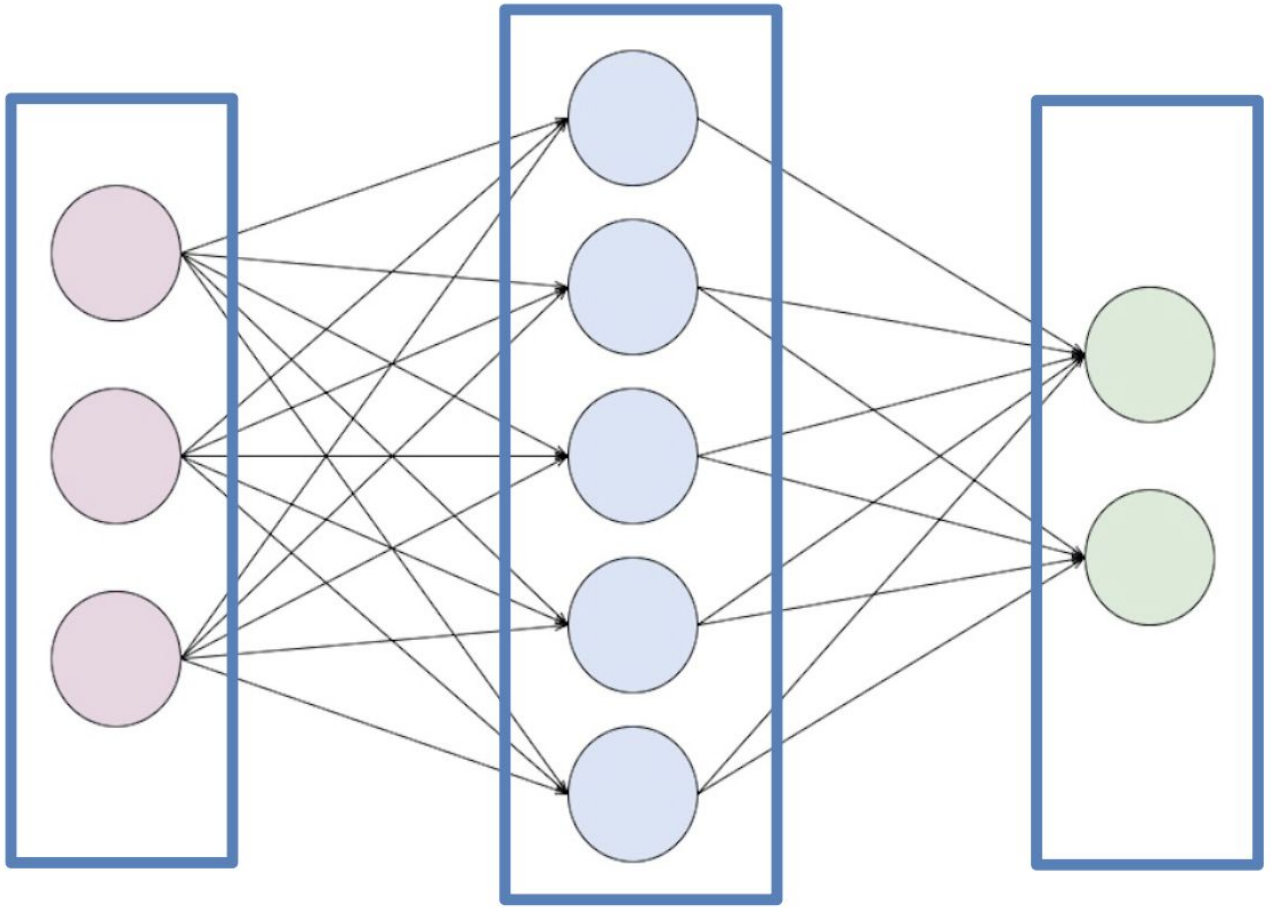
Disclaimer:

Several names:

- Dense
- Feed-forward
- NNs
- MLP
- ANNs
- DNNs
- …



DNNs



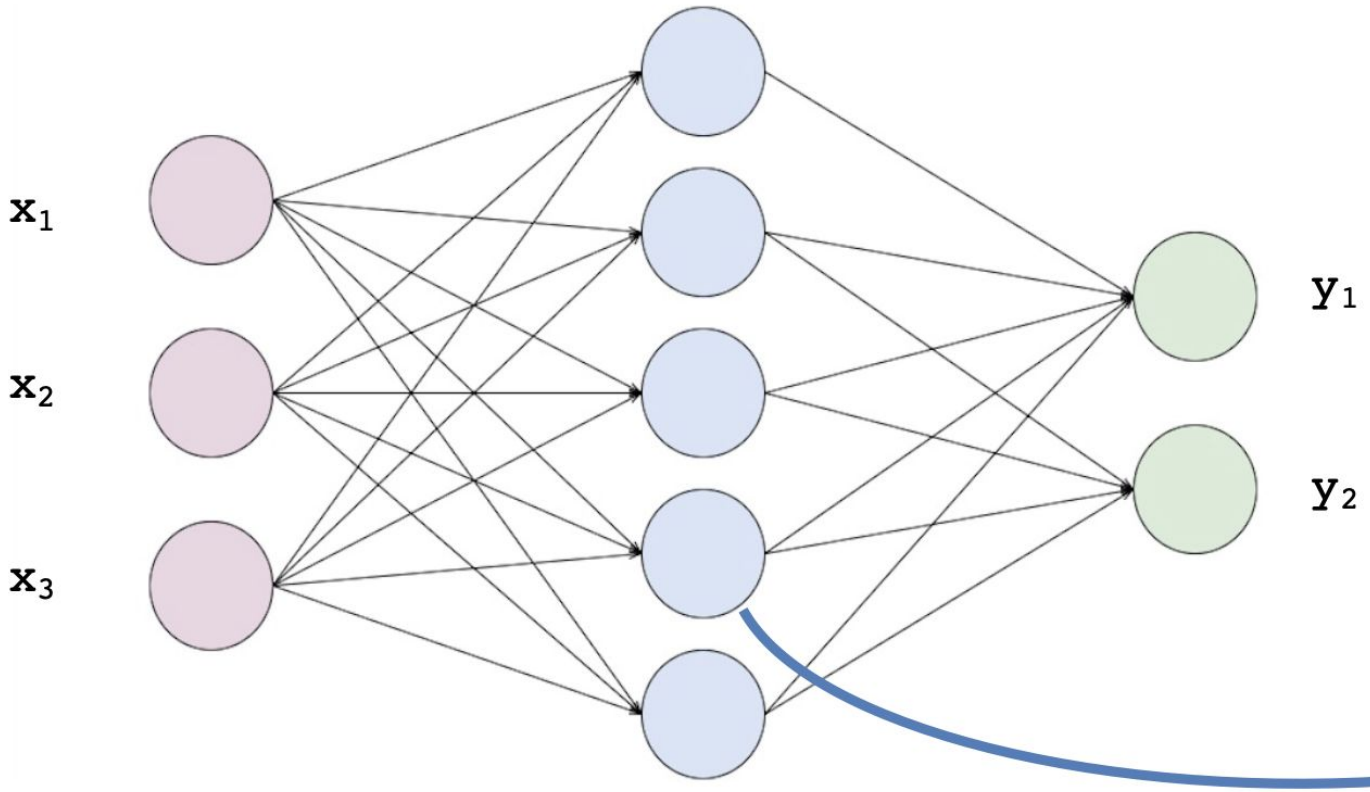
The Input Layer

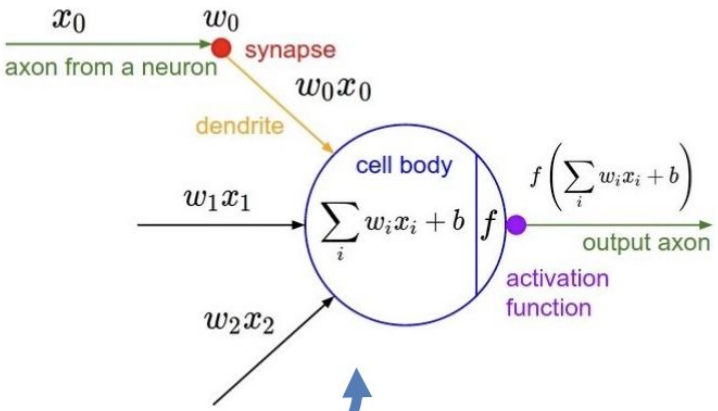
The Hidden Layer

The Output Layer

DNNs

$$Y = F(x)$$


x1
x2
x3
y1
y2


x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$w_1 x_1$
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation function
$w_2 x_2$

ANNs. Keras

```
my_first_neural_net = tf.keras.Sequential([
    layers.Input(shape=(3, ), name="input"),
    layers.Dense(units=5, activation="sigmoid", name="hidden"),
    layers.Dense(units=2, activation="softmax", name="output")
])
```

ANNs. Keras

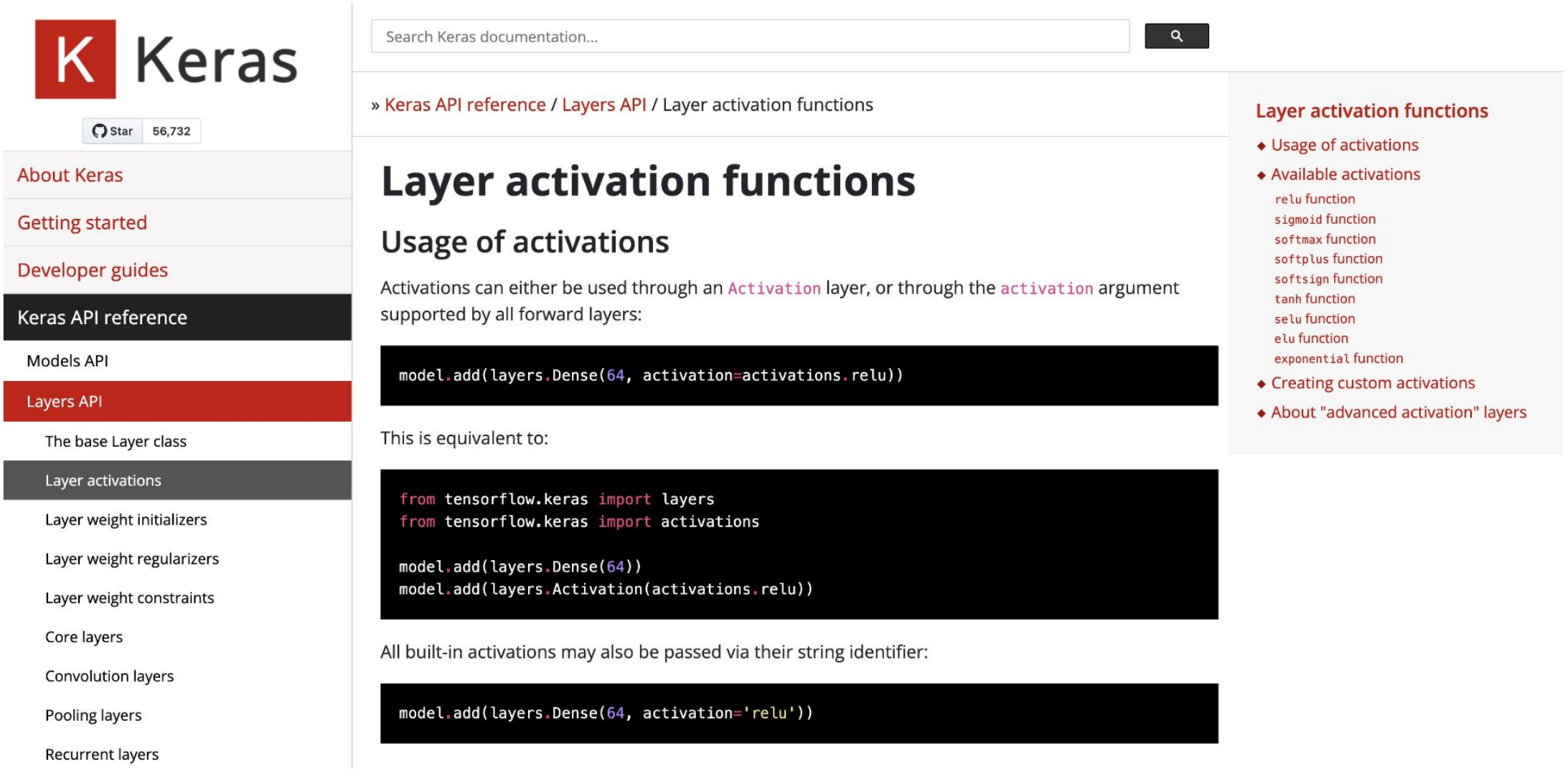
```
my_first_neural_net.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 hidden (Dense)              (None, 5)                 20

 output (Dense)              (None, 2)                 12

=================================================================
Total params: 32
Trainable params: 32
Non-trainable params: 0
_________________________________________________________________
```

ANNs. Activation Functions



Keras

Search Keras documentation...

Star 56,732

- About Keras
- Getting started
- Developer guides
- Keras API reference
  - Models API
  - Layers API
    - The base Layer class
    - Layer activations
    - Layer weight initializers
    - Layer weight regularizers
    - Layer weight constraints
    - Core layers
    - Convolution layers
    - Pooling layers
    - Recurrent layers

» Keras API reference / Layers API / Layer activation functions

Layer activation functions

Usage of activations

Activations can either be used through an `Activation` layer, or through the `activation` argument supported by all forward layers:

```
model.add(layers.Dense(64, activation=activations.relu))
```

This is equivalent to:

```
from tensorflow.keras import layers
from tensorflow.keras import activations

model.add(layers.Dense(64))
model.add(layers.Activation(activations.relu))
```

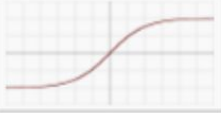
All built-in activations may also be passed via their string identifier:

```
model.add(layers.Dense(64, activation='relu'))
```

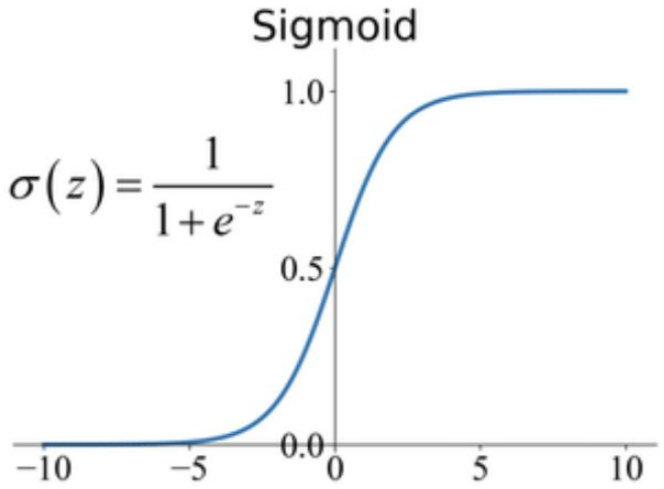
Layer activation functions

- Usage of activations
- Available activations
  - `relu` function
  - `sigmoid` function
  - `softmax` function
  - `softplus` function
  - `softsign` function
  - `tanh` function
  - `selu` function
  - `elu` function
  - `exponential` function
- Creating custom activations
- About "advanced activation" layers

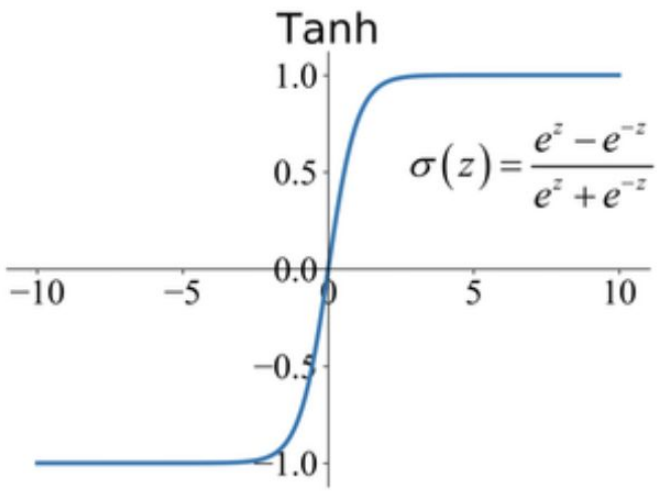
ANNs. Activation Functions

Name	Plot	Equation
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$

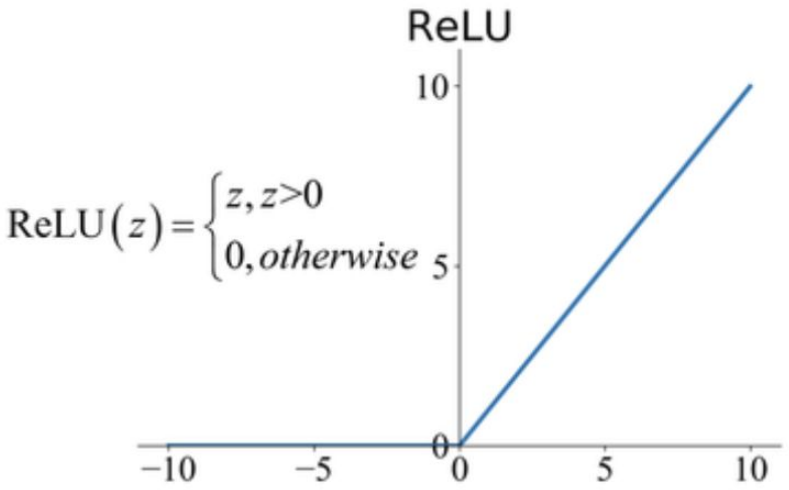
ANNs. Activation Functions



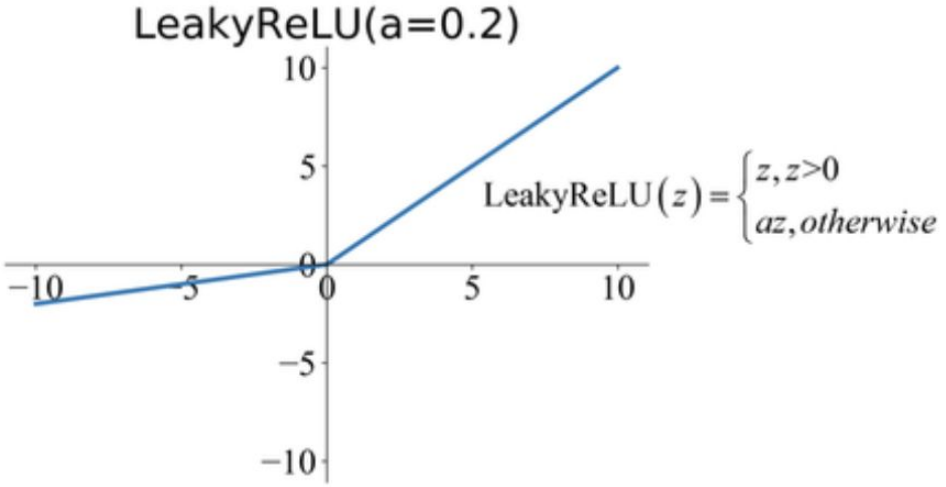
(a)



(b)



(c)



(d)

ANNs. Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

ANNs. Activation Functions

- The combination of linear functions is just another linear function. No matter the number of hidden layers, if we use linear functions as activations, we could just deal with outputs that are a linear combination of our inputs... not interesting.

- Non-linear activations functions are key in neural nets to represent non-linear mapping functions.

- **In the output layer**, at general: use sigmoid for binary classification, softmax for multiclass classification and linear for regression.

    - You can use softmax for binary classification using two outputs and in the case of a problem where data that can belong to multiple classes, then use sigmoid. [i.e., a picture of Audi belongs to the classes' car and vehicle]

- **In hidden layers:**
  - Tanh use to work better than sigmoid as it centers data.
  - ReLu is faster and it works very well too. It is the most used.