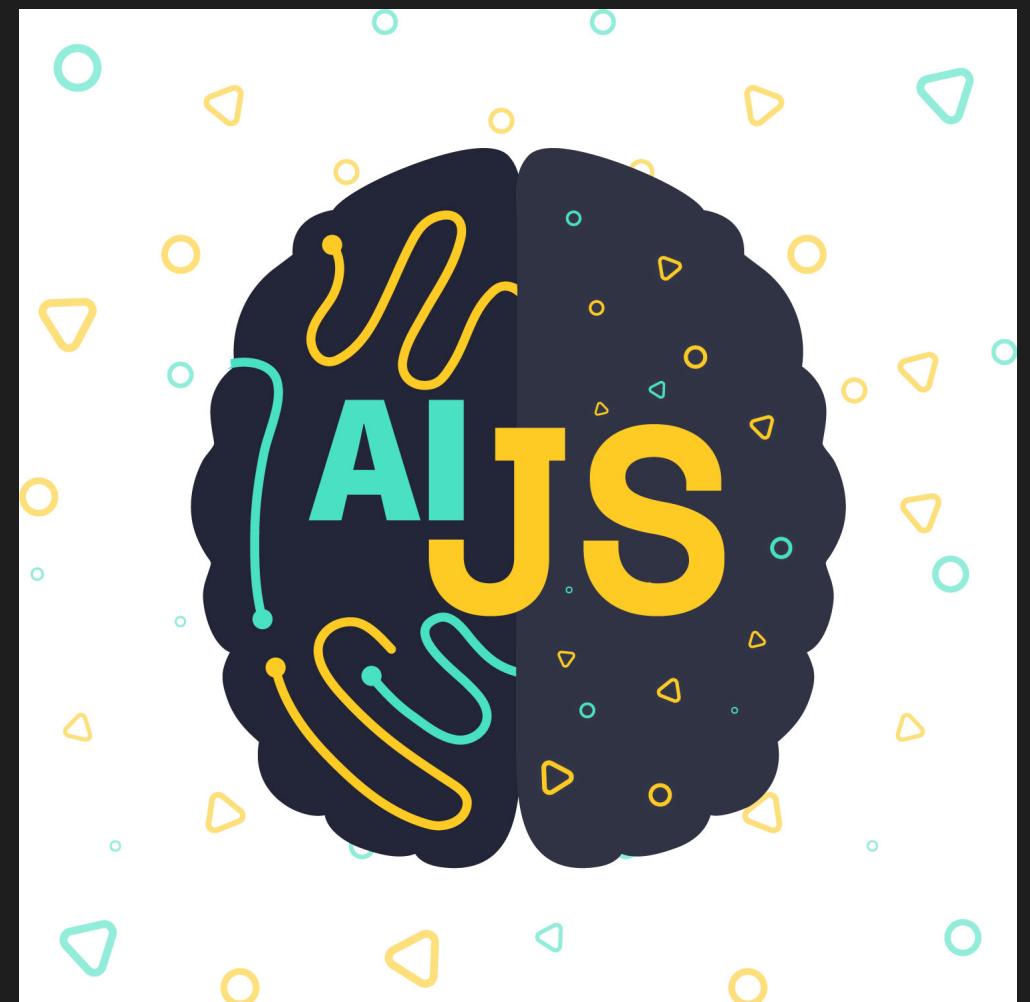


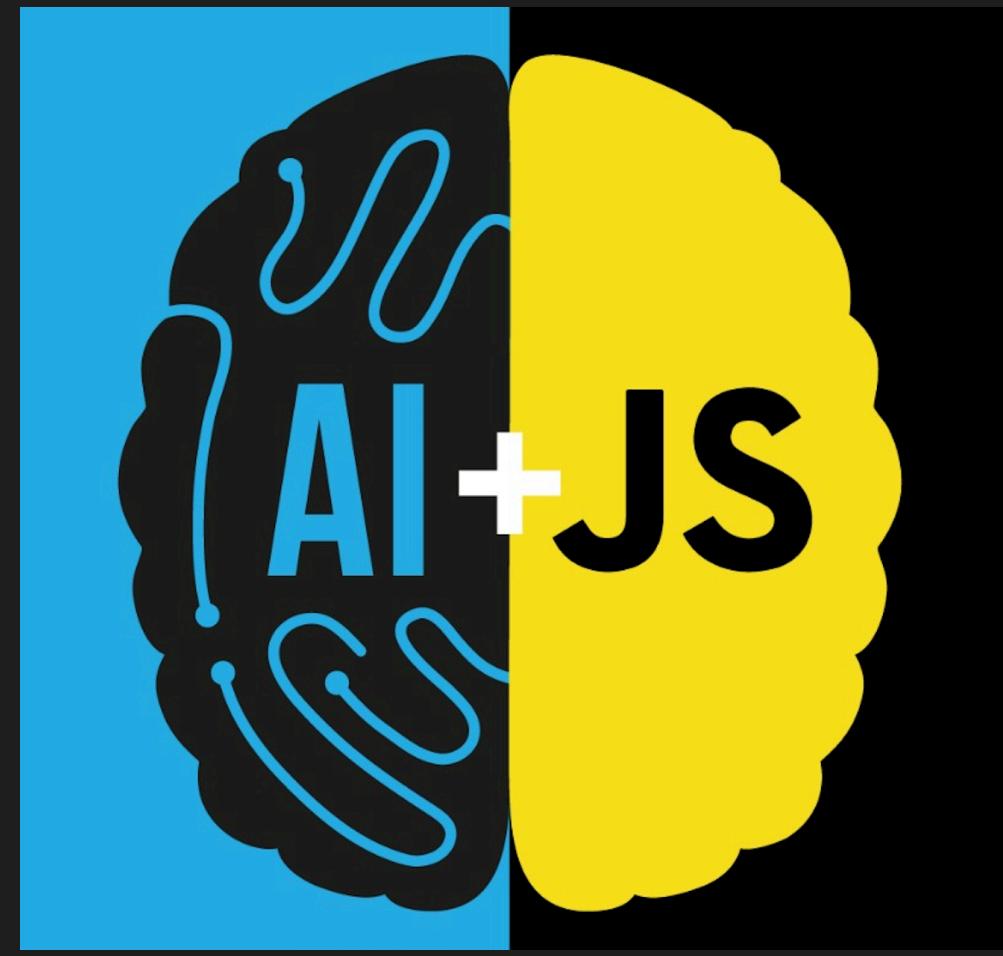
@EleanorHaproff



@MarvelApp



aijs.rocks



AI JavaScript London  
@aijavascript

# Machine

# Learning &

# JavaScript

What is AI, Machine  
Learning & Data Science?

**Solving complex problems  
using data**

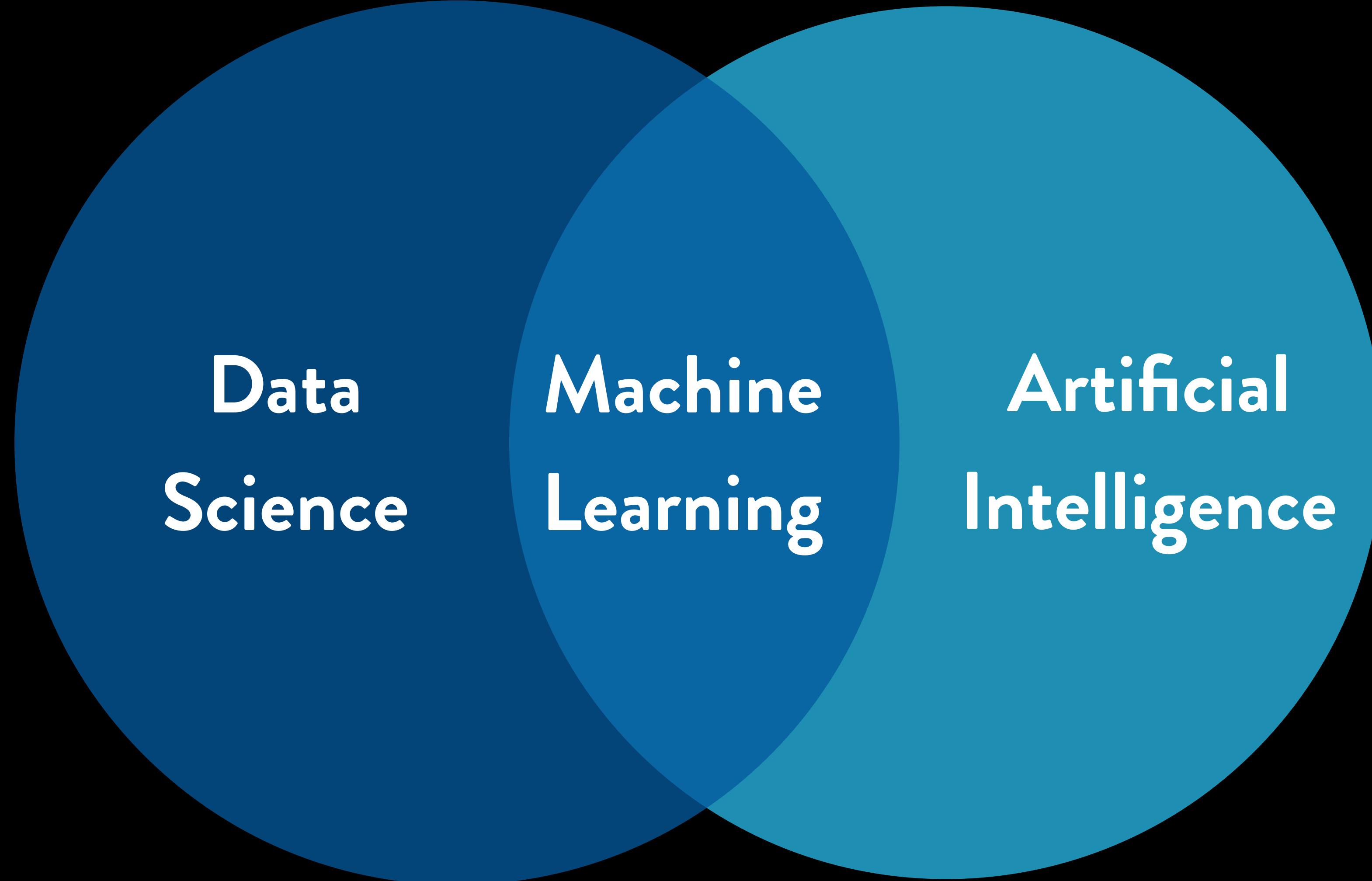
**Data  
Science**

**Simulation of a human  
brain function by machines**

**Machine  
Learning**

**Artificial  
Intelligence**

**Learning from data**



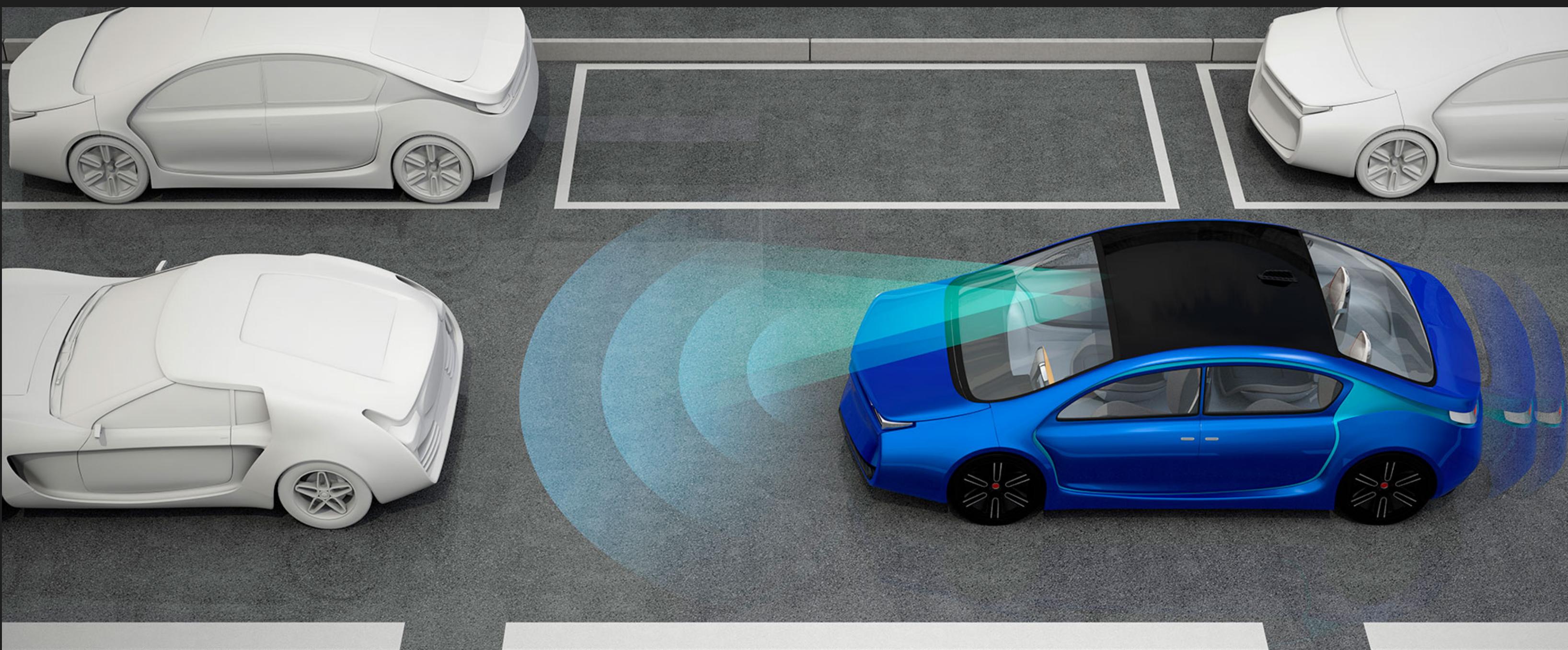
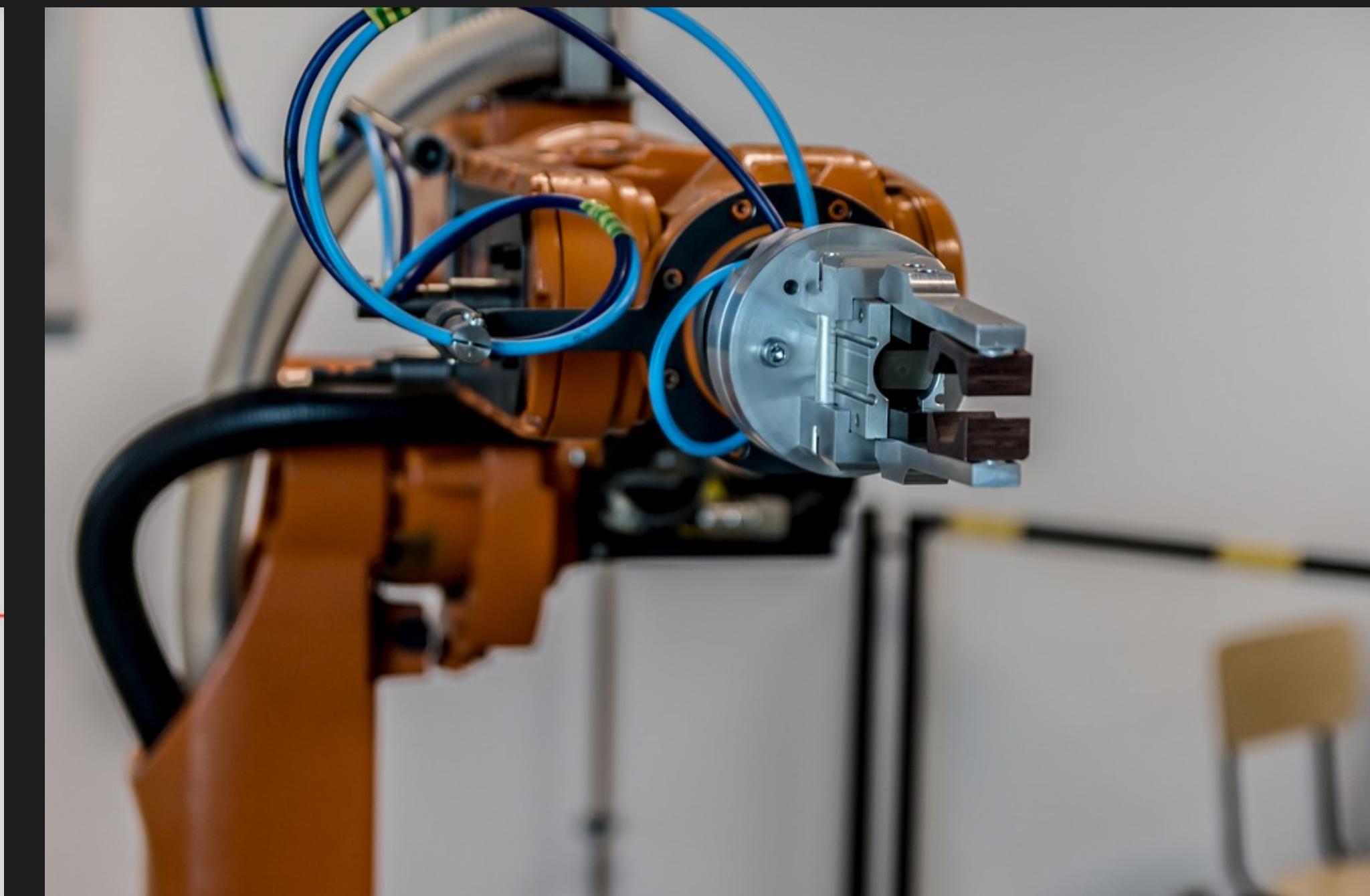
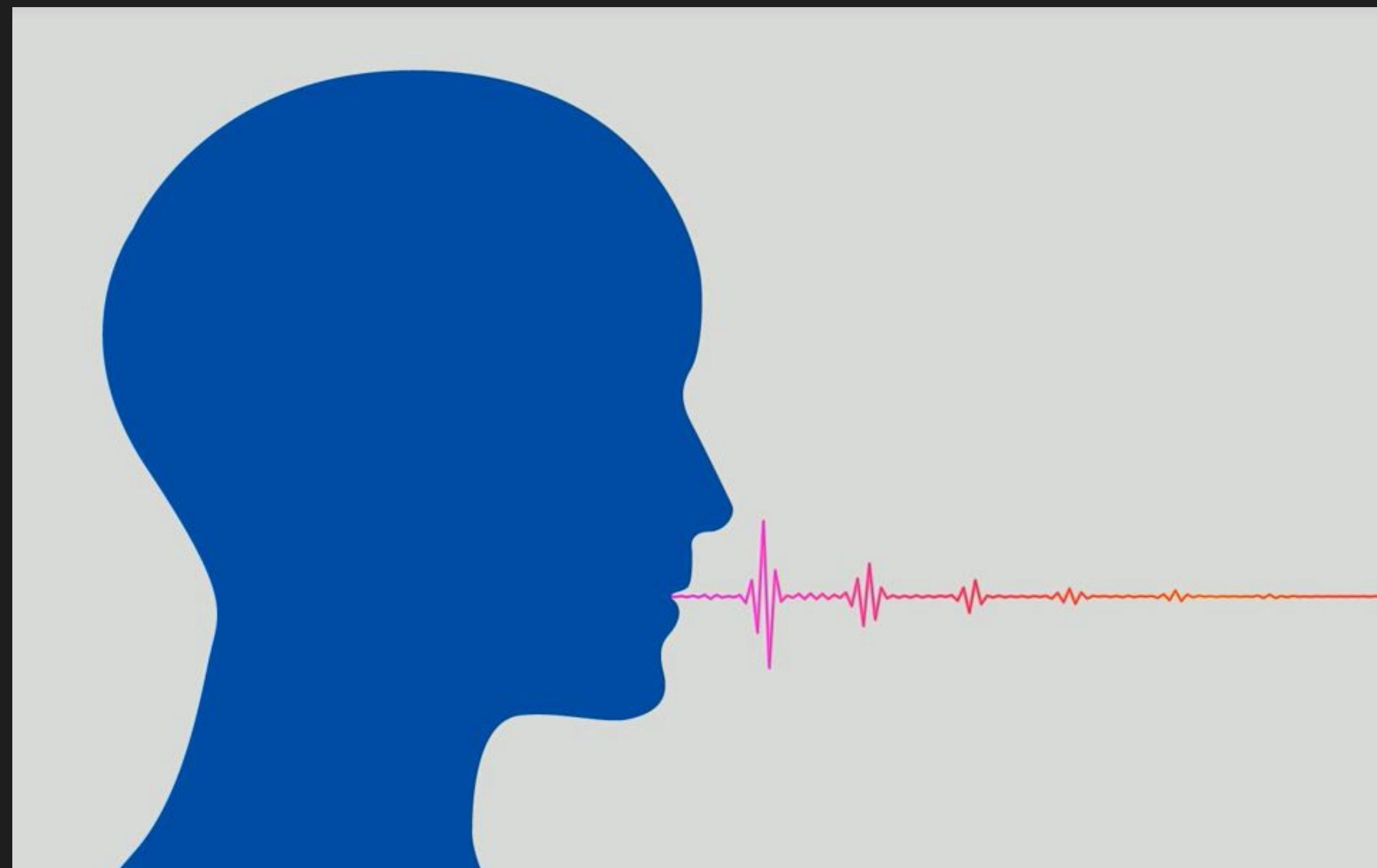
A Venn diagram consisting of three overlapping circles. The leftmost circle is dark blue and contains the white text "Data Science". The middle circle is medium blue and contains the white text "Machine Learning". The rightmost circle is light blue and contains the white text "Artificial Intelligence". The overlapping areas between the circles represent the intersections of these fields.

Data  
Science

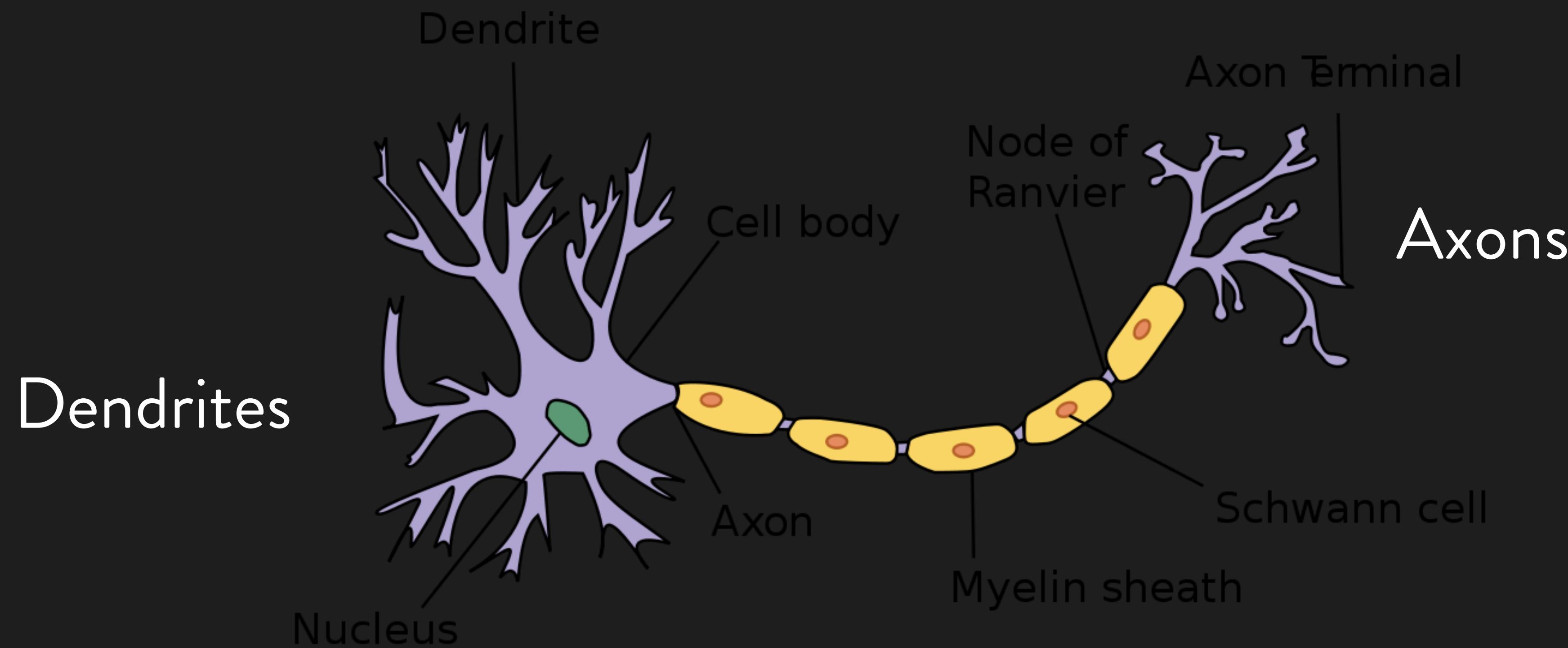
Machine  
Learning

Artificial  
Intelligence

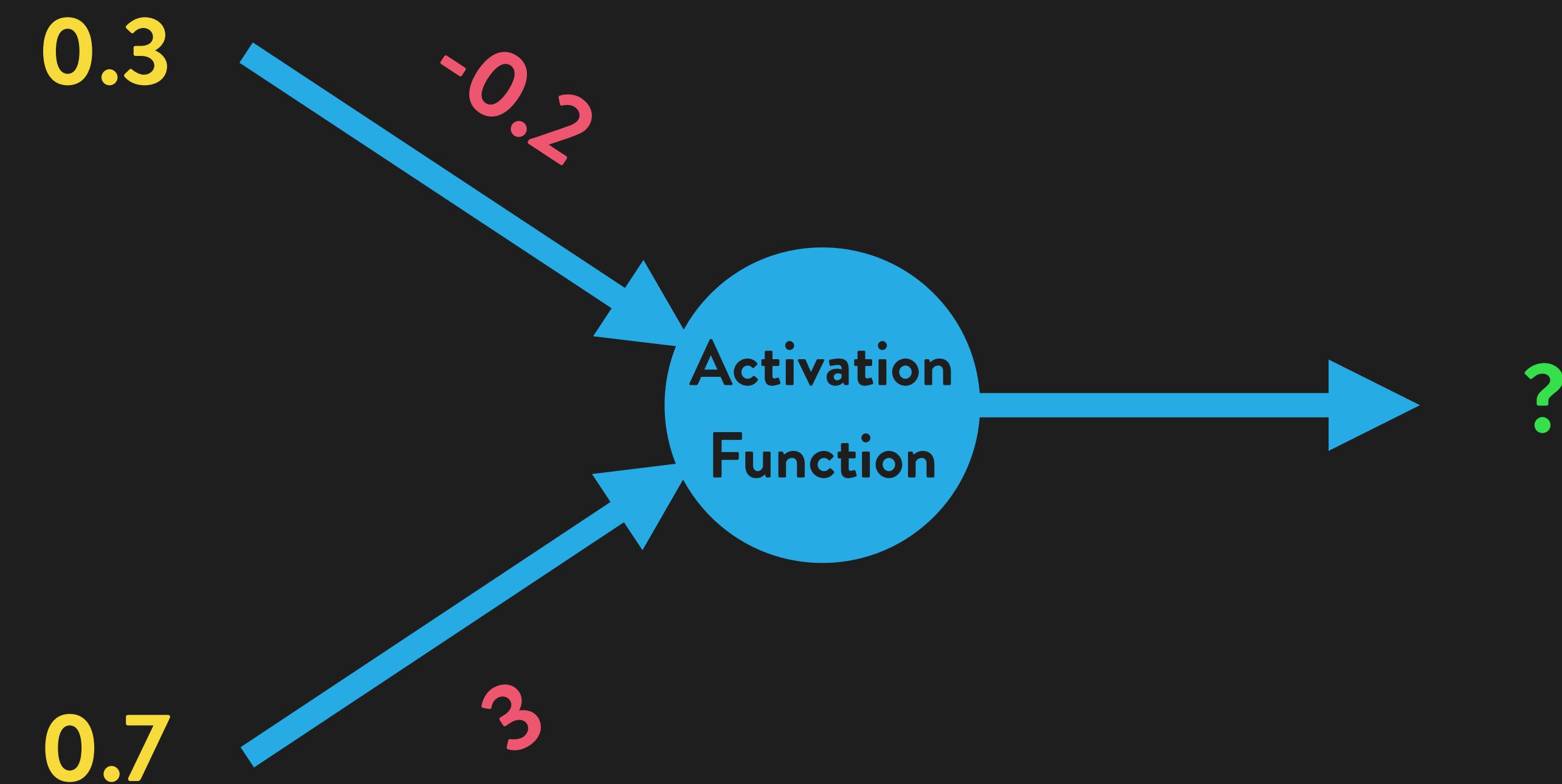
# Artificial Neural Networks



# Biological Neuron



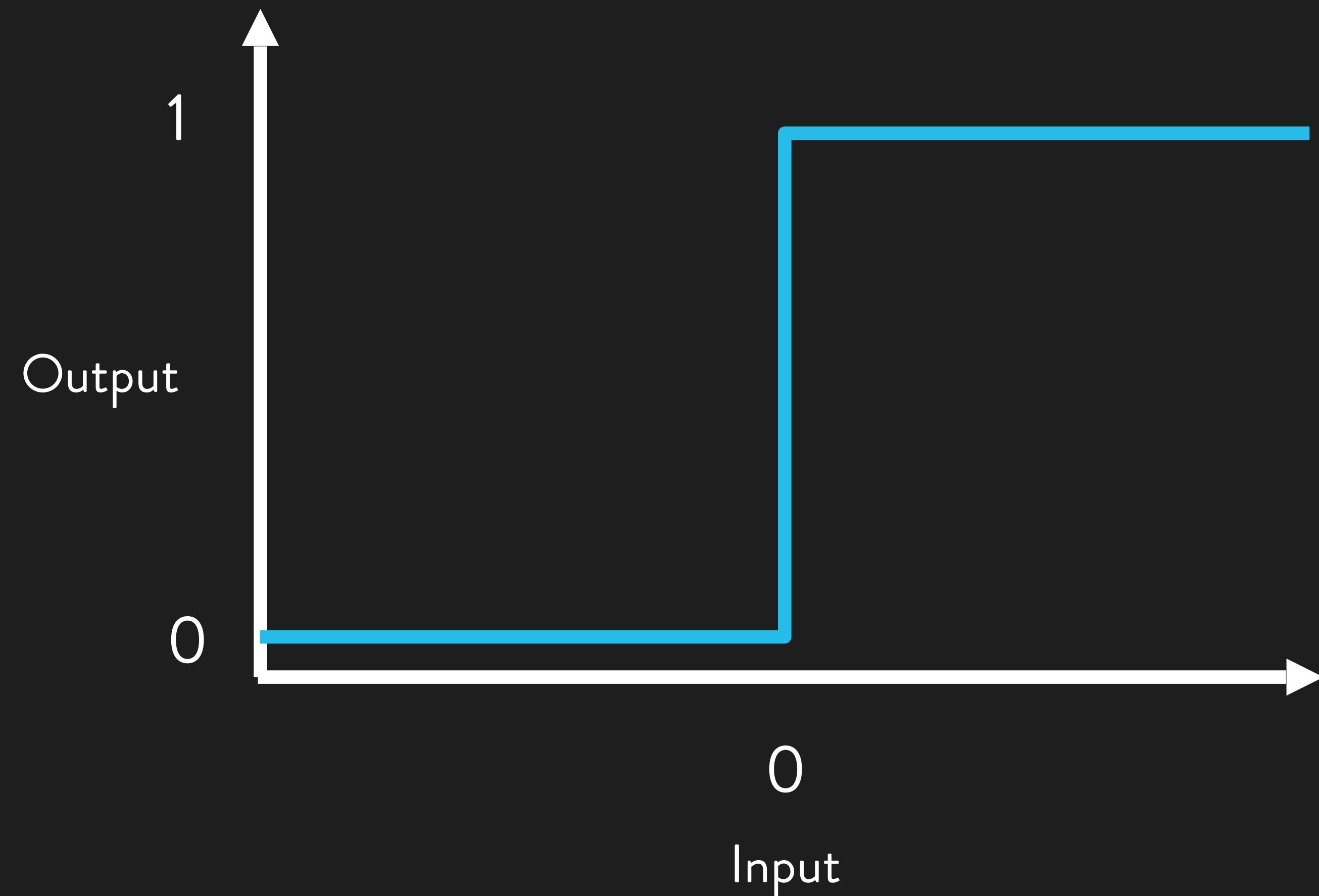
# Artificial Neuron



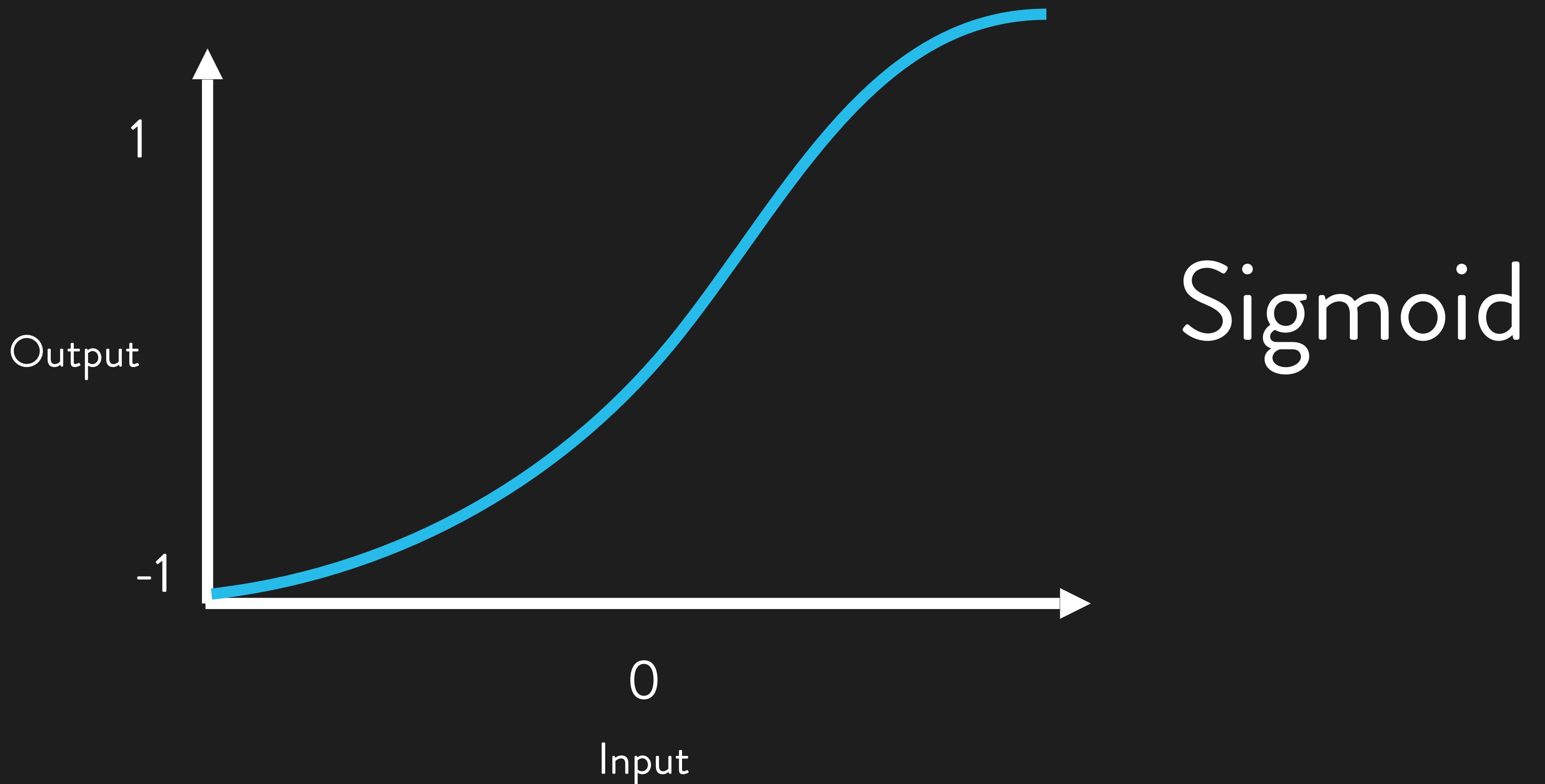
# Artificial Neuron

$$\left. \begin{array}{l} 0.3 \times -0.2 = -0.6 \\ 0.7 \times 3 = 2.1 \end{array} \right\} 2.7 \rightarrow \text{activation}(\dots) \rightarrow \uparrow$$

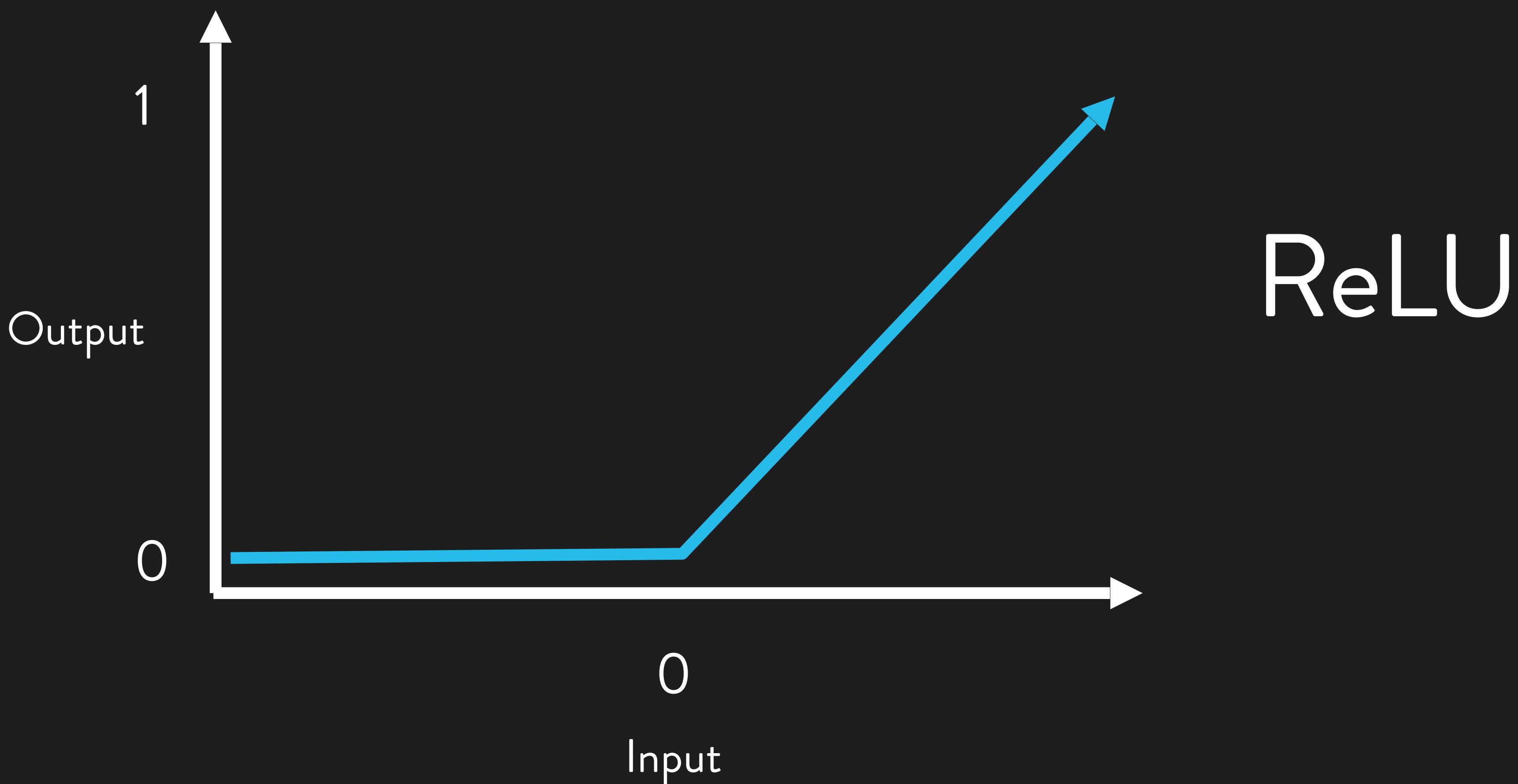
# Activation functions



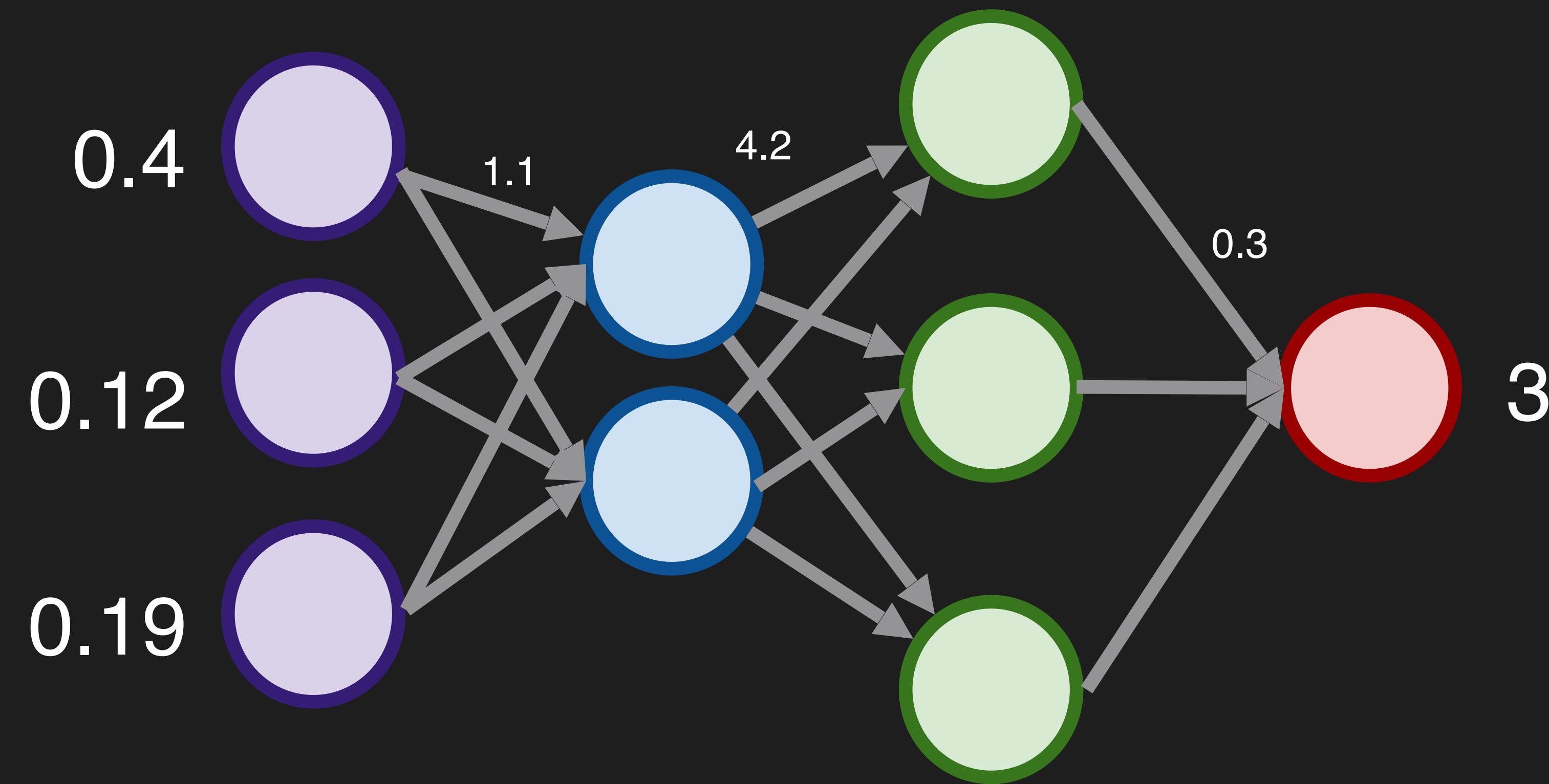
# Activation functions



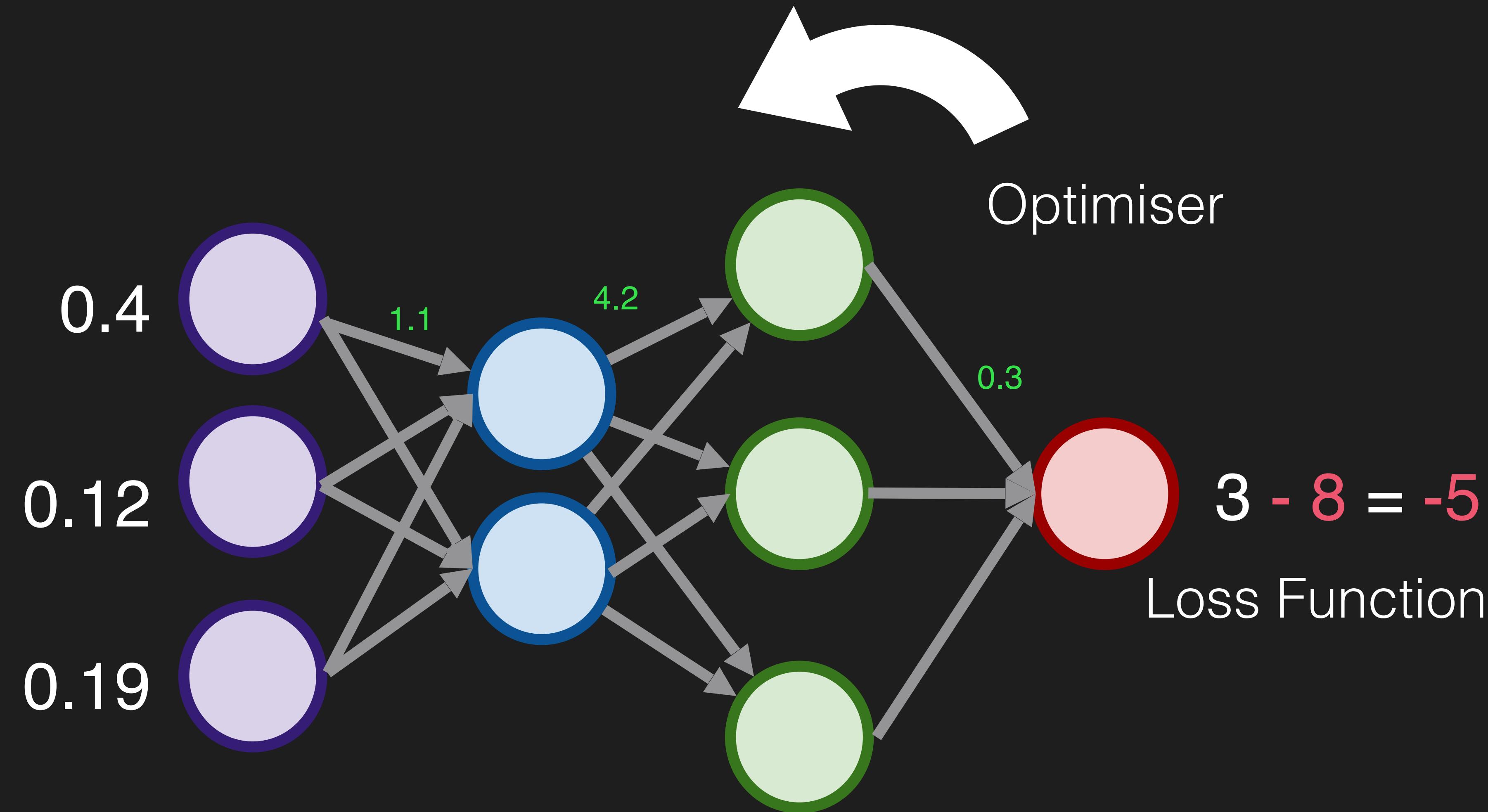
# Activation functions



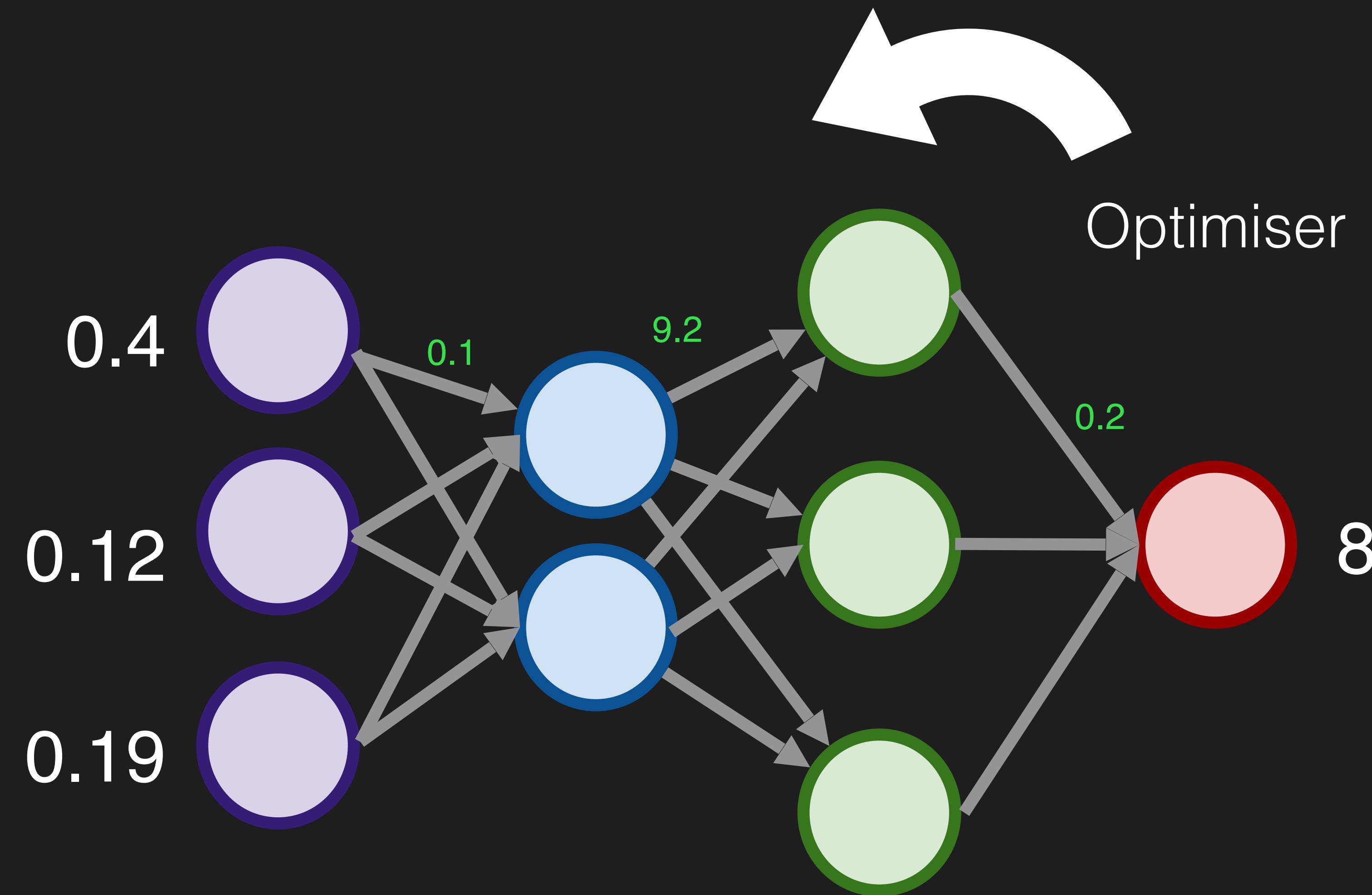
# Artificial Neural Networks



# Artificial Neural Networks



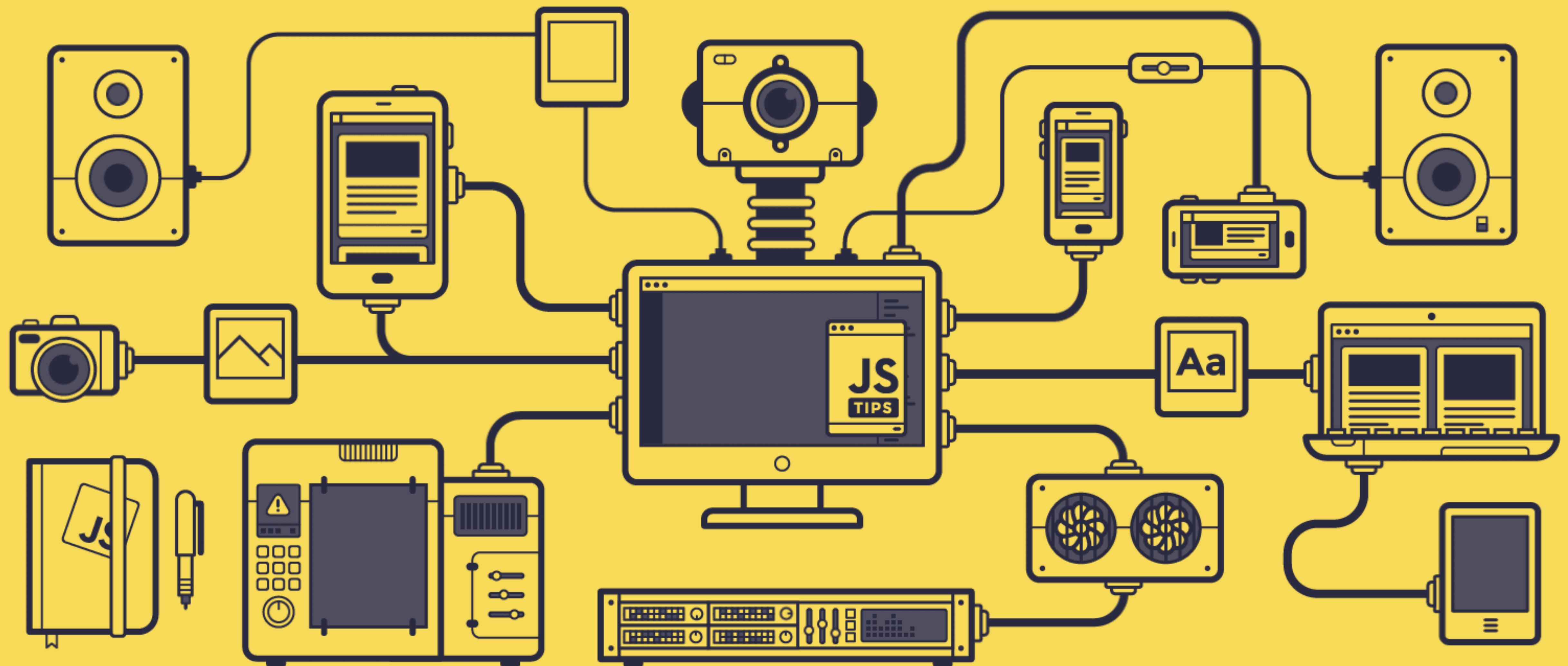
# Artificial Neural Networks



Existing  
Landscape  
of ML

MACHINE  
LEARNING





<http://www.jstips.co/en/about/>

# Why JavaScript?

stack overflow

Overview  
Developer Profile  
**Technology**  
**I. Most Popular Technologies**  
II. Most Loved, Dreaded, and Wanted  
III. Development Environments and Tools  
IV. Top Paying Technologies  
V. Correlated Technologies  
VI. Technology and Society  
Work  
Community  
Methodology

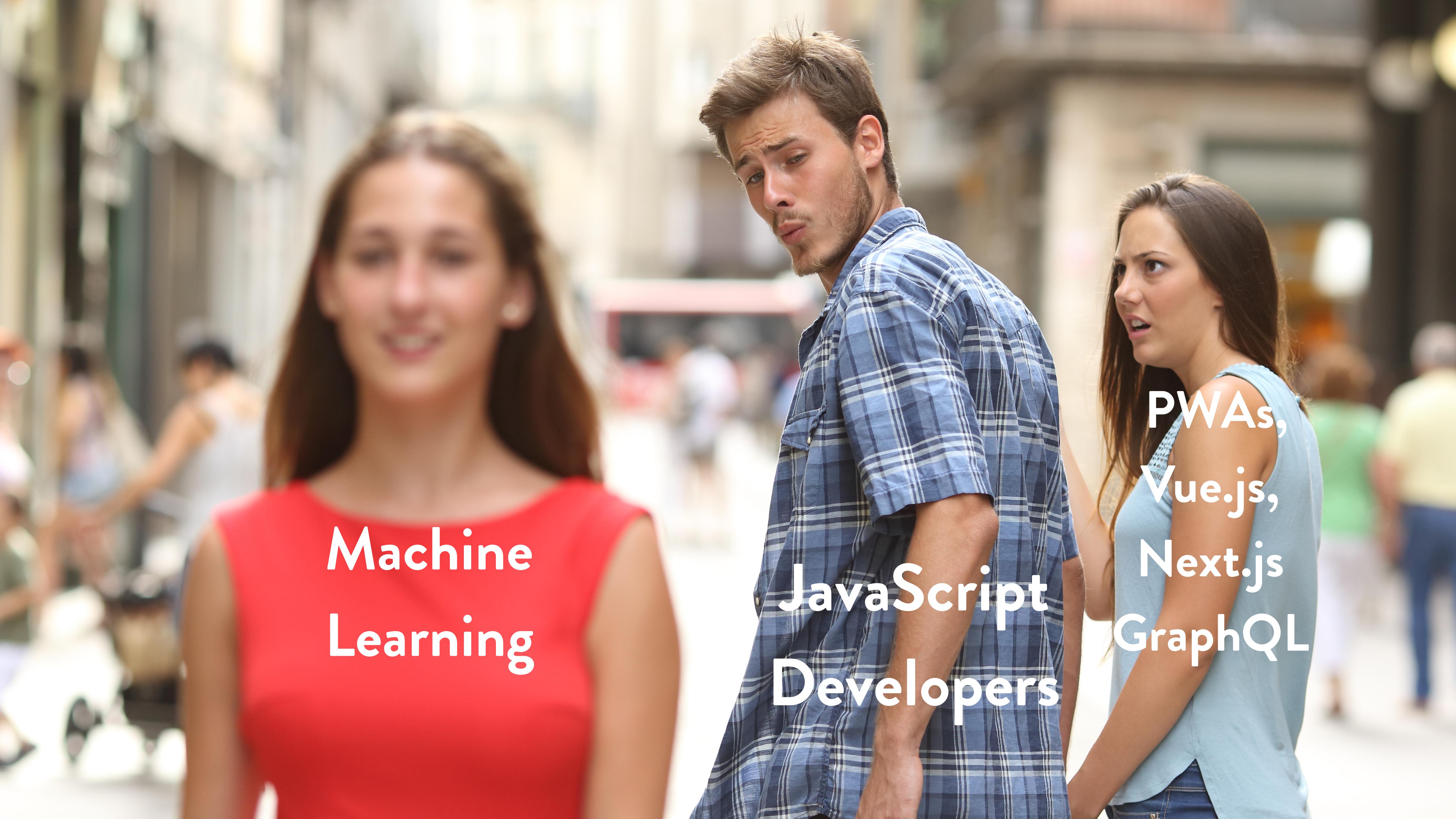
Most Popular Technologies

### Programming, Scripting, and Markup Languages

All Respondents Professional Developers

A horizontal bar chart titled 'Programming, Scripting, and Markup Languages' comparing the popularity of various technologies across 'All Respondents' and 'Professional Developers'. The chart uses orange bars to represent the percentage of users for each technology. The technologies listed are: JavaScript, HTML, CSS, SQL, Java, Bash/Shell, Python, C#, PHP, C++, C, TypeScript, Ruby, Swift, Objective-C, and Go. JavaScript is the most popular at 71.5% for all respondents.

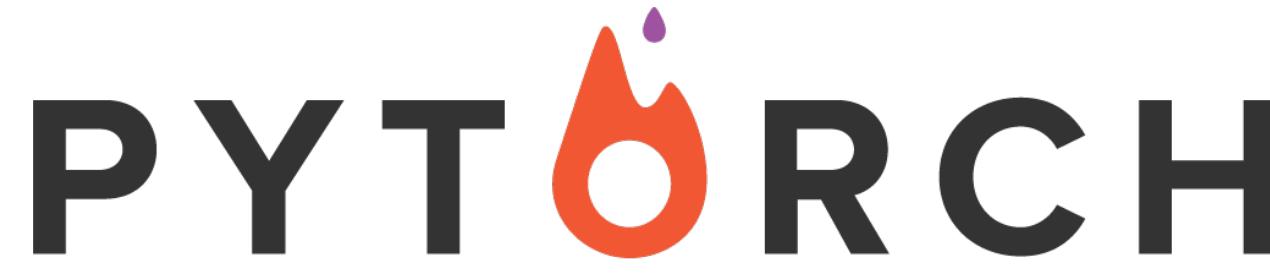
| Technology  | All Respondents (%) | Professional Developers (%) |
|-------------|---------------------|-----------------------------|
| JavaScript  | 71.5%               |                             |
| HTML        | 69.4%               |                             |
| CSS         | 66.2%               |                             |
| SQL         | 58.5%               |                             |
| Java        | 45.4%               |                             |
| Bash/Shell  | 40.4%               |                             |
| Python      | 37.9%               |                             |
| C#          | 35.3%               |                             |
| PHP         | 31.4%               |                             |
| C++         | 24.6%               |                             |
| C           | 22.1%               |                             |
| TypeScript  | 18.3%               |                             |
| Ruby        | 10.3%               |                             |
| Swift       | 8.3%                |                             |
| Objective-C | 7.3%                |                             |
| Go          | 7.2%                |                             |



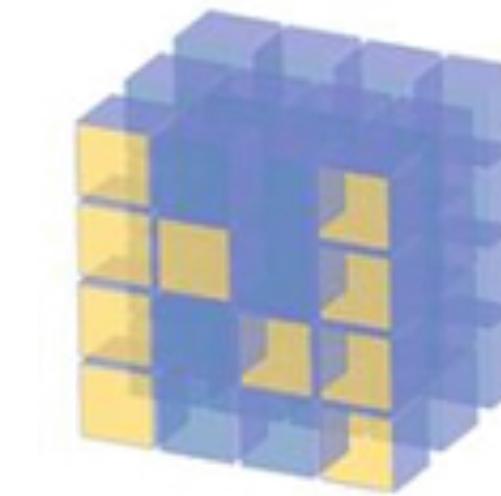
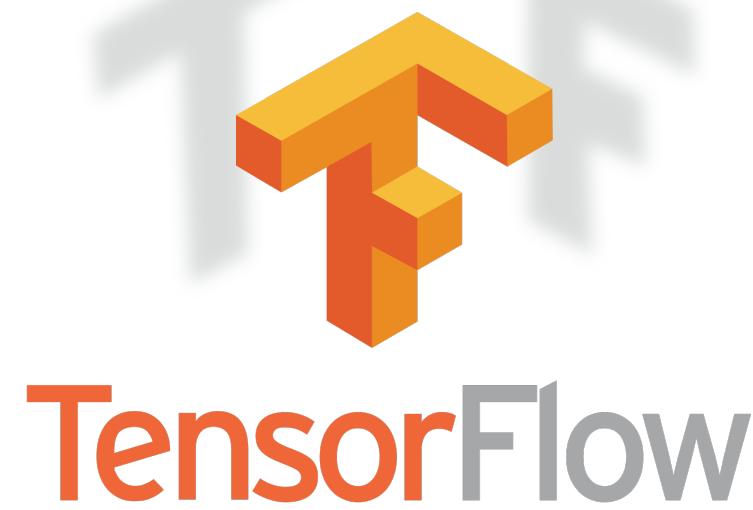
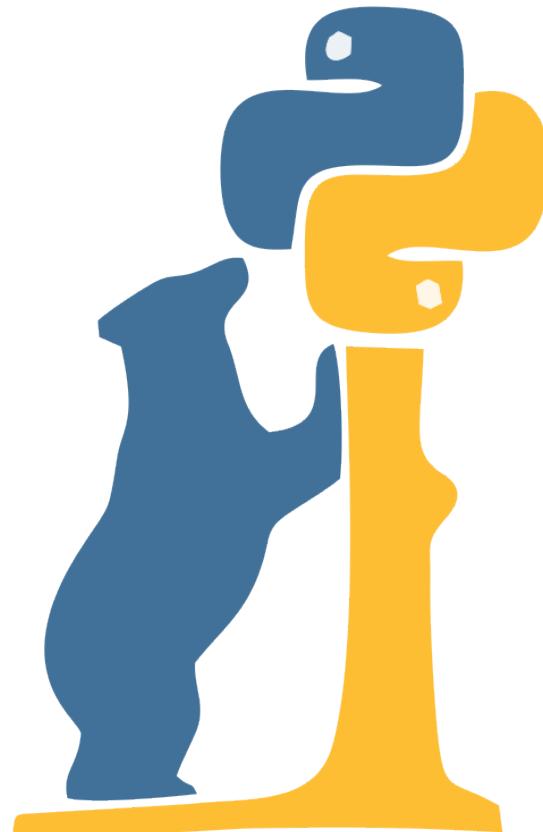
Machine  
Learning

JavaScript  
Developers

PWAs,  
Vue.js,  
Next.js  
GraphQL



Pandas



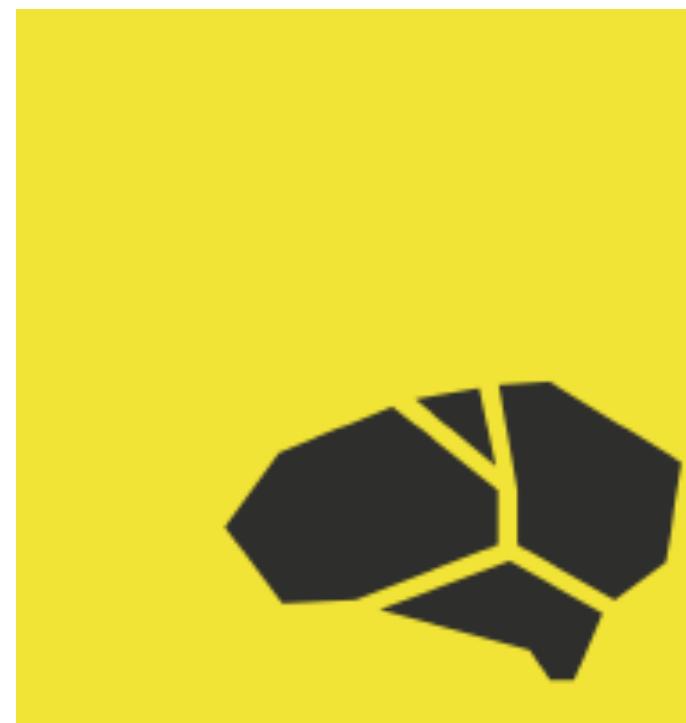
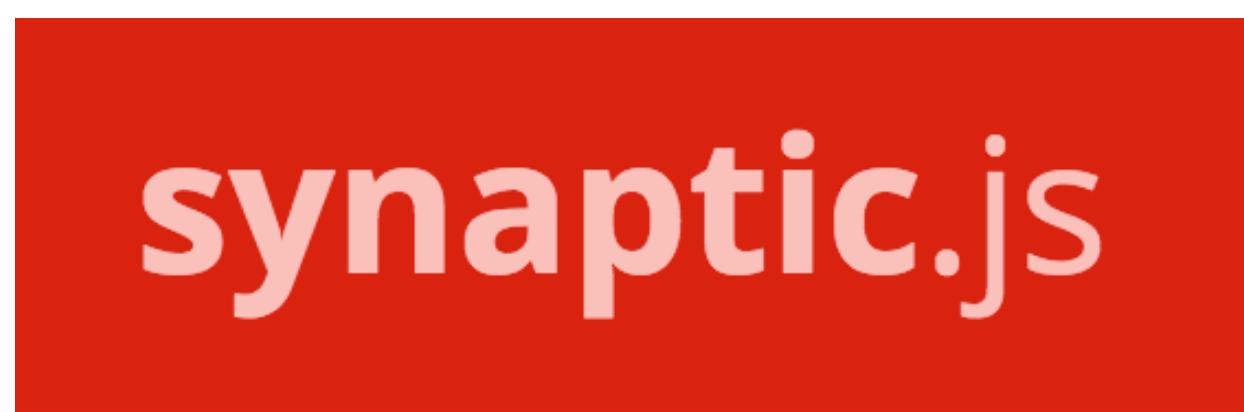
NumPy



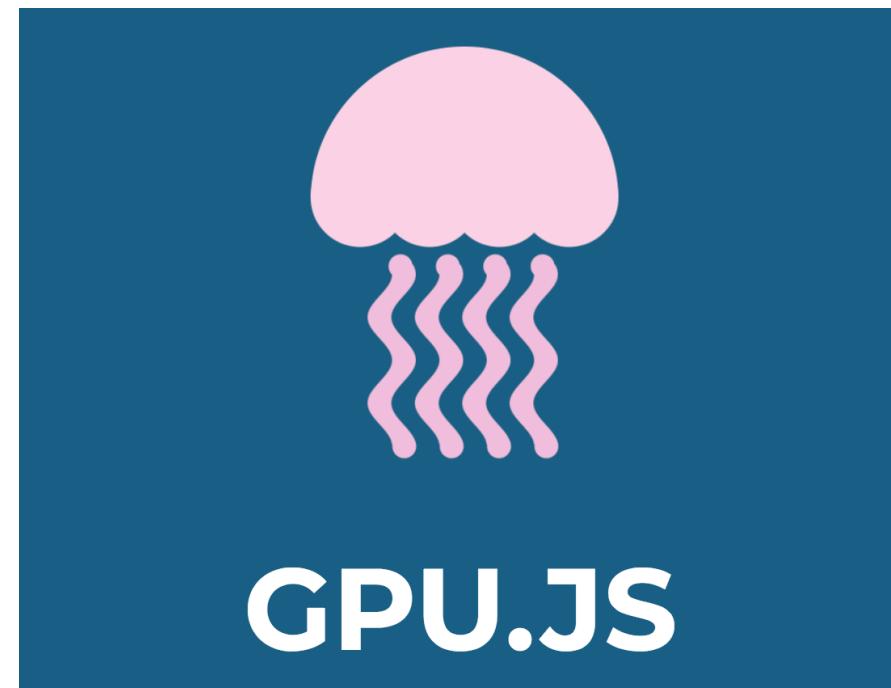
spaCy  
theano



# JavaScript and Machine Learning



● Propel



math.js



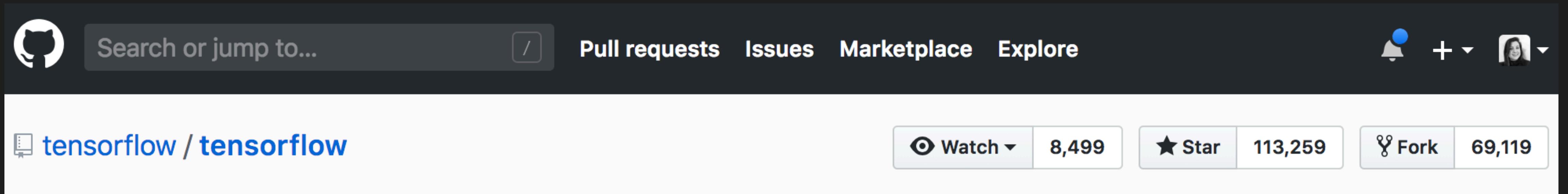
# TensorFlow.js

# TensorFlow

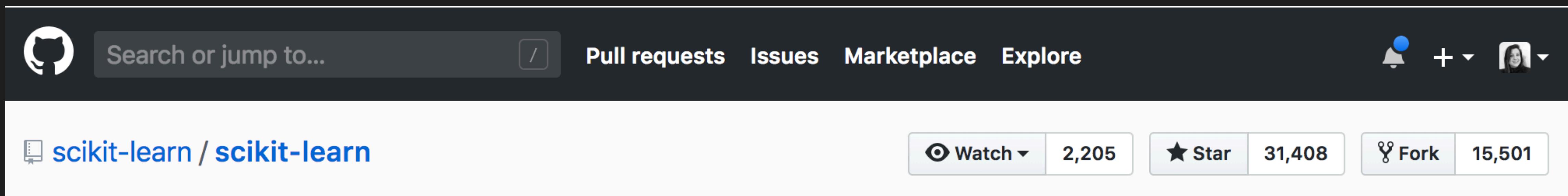


# TensorFlow

# TensorFlow



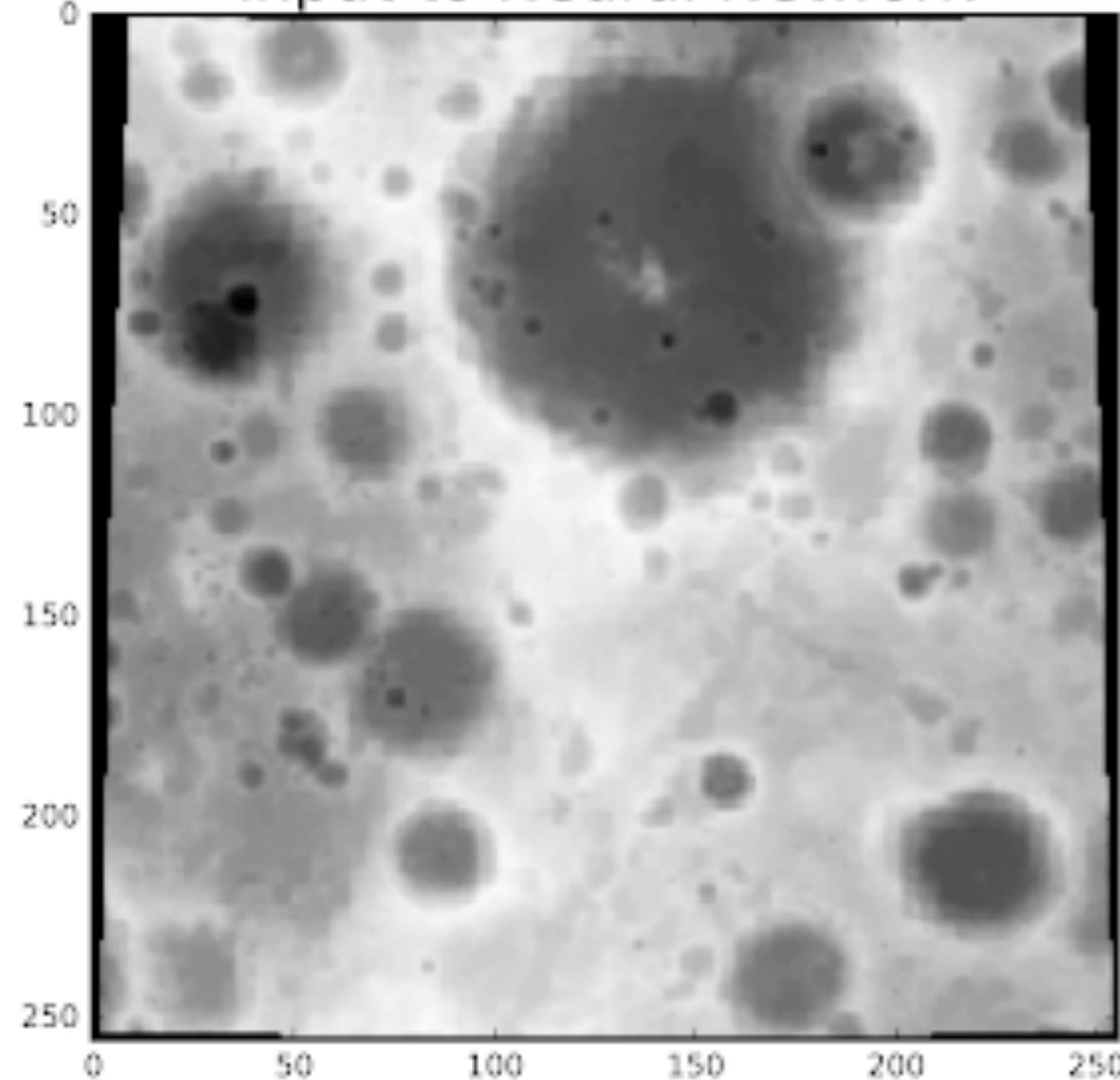
A screenshot of the TensorFlow GitHub repository page. The page has a dark header with the GitHub logo, a search bar, and navigation links for Pull requests, Issues, Marketplace, and Explore. On the right, there are icons for notifications, a plus sign, and a user profile. Below the header, the repository name "tensorflow / tensorflow" is displayed, along with a "Watch" button (8,499), a "Star" button (113,259), and a "Fork" button (69,119).



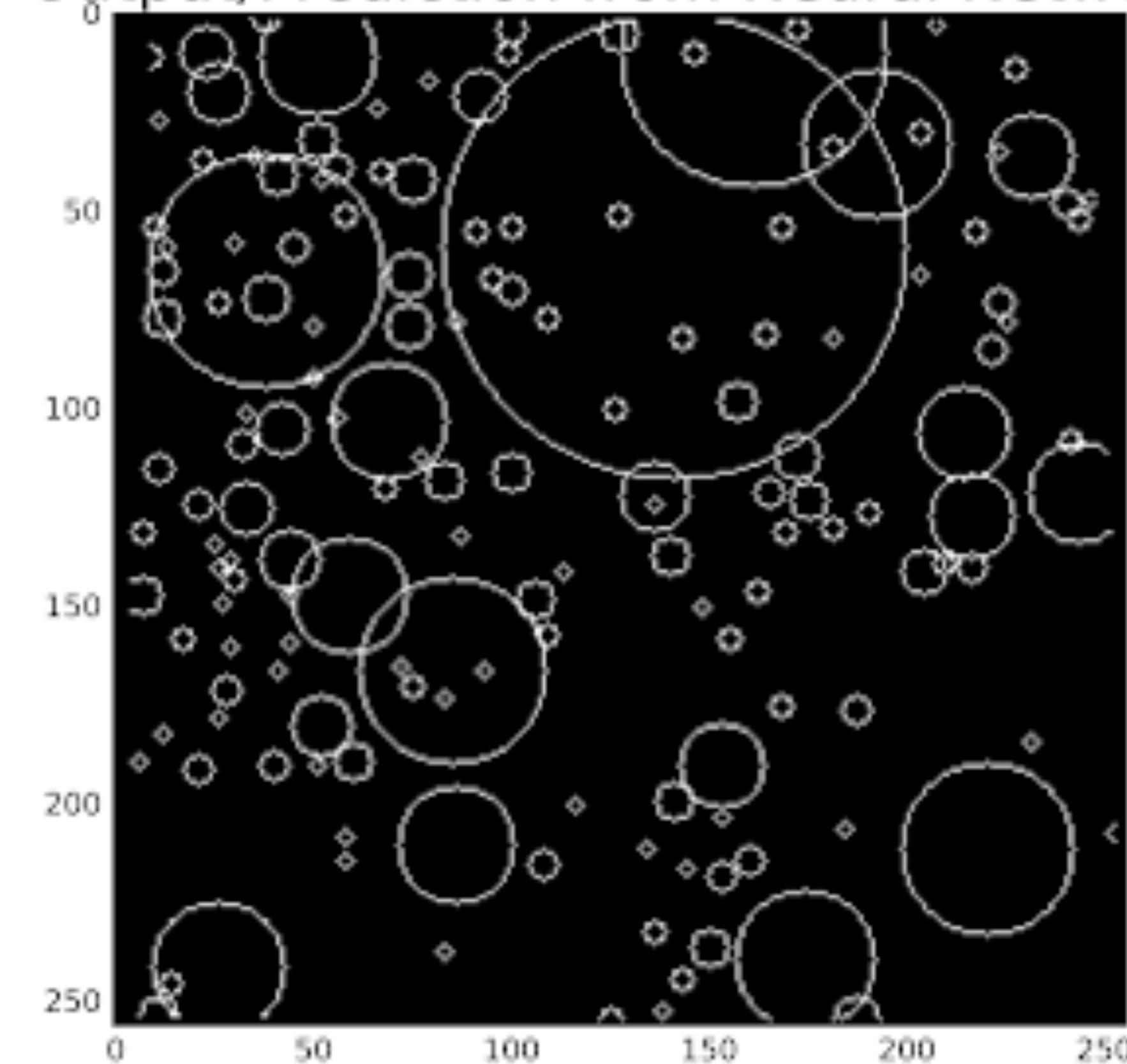
A screenshot of the scikit-learn GitHub repository page. The layout is identical to the TensorFlow page, featuring a dark header with the GitHub logo, a search bar, and navigation links. The repository name "scikit-learn / scikit-learn" is shown, along with a "Watch" button (2,205), a "Star" button (31,408), and a "Fork" button (15,501).

# TensorFlow

Input to Neural Network



Output/Prediction from Neural Network





# TensorFlow.js



# TensorFlow Playground

<https://playground.tensorflow.org/>

Epoch 000,034    Learning rate 0.03    Activation Tanh    Regularization None    Regularization rate 0    Problem type Classification

DATA    FEATURES    2 HIDDEN LAYERS    OUTPUT

Which dataset do you want to use?

Which properties do you want to feed in?

+ -

2 neurons

2 neurons

$x_1$      $x_2$      $x_1^2$      $x_2^2$      $x_1x_2$      $\sin(x_1)$

$x_1$      $x_2$      $x_1^2$      $x_2^2$      $x_1x_2$      $\sin(x_1)$

+ -

This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

Test loss 0.460  
Training loss 0.470

Ratio of training to test data: 50%

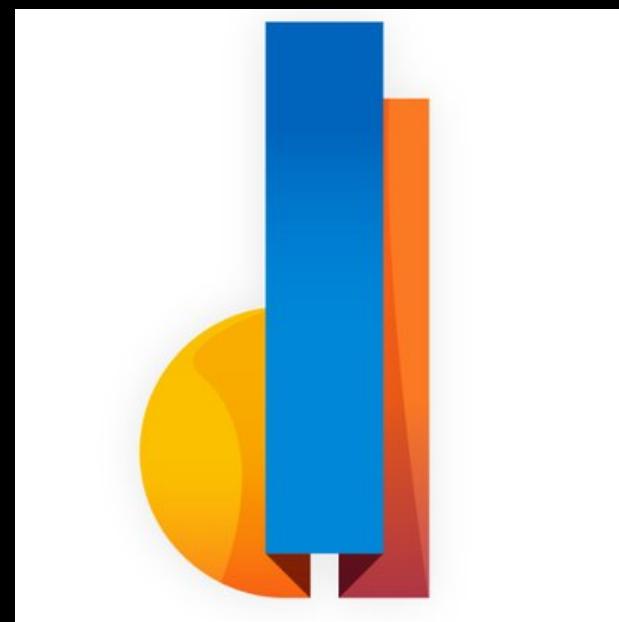
Noise: 0

Batch size: 10

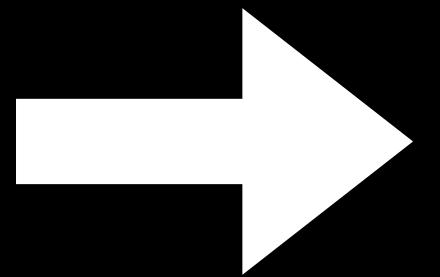
REGENERATE

The TensorFlow Playground interface displays a neural network configuration with two hidden layers, each containing two neurons. The input features are  $x_1$ ,  $x_2$ ,  $x_1^2$ ,  $x_2^2$ ,  $x_1x_2$ , and  $\sin(x_1)$ . The output shows a scatter plot of data points colored by class (blue and orange) and overlaid with a decision boundary and confidence contours. The network's performance is indicated by a test loss of 0.460 and a training loss of 0.470. Various controls at the bottom left allow for data regeneration, setting batch sizes, and adjusting noise levels.

# TensorFlow.js



deeplearn.js



 TensorFlow.js

The TensorFlow.js logo, which consists of a stylized orange 'F' icon followed by the word "TensorFlow.js" in orange and grey text.

# deeplearn.js & Style Transfer

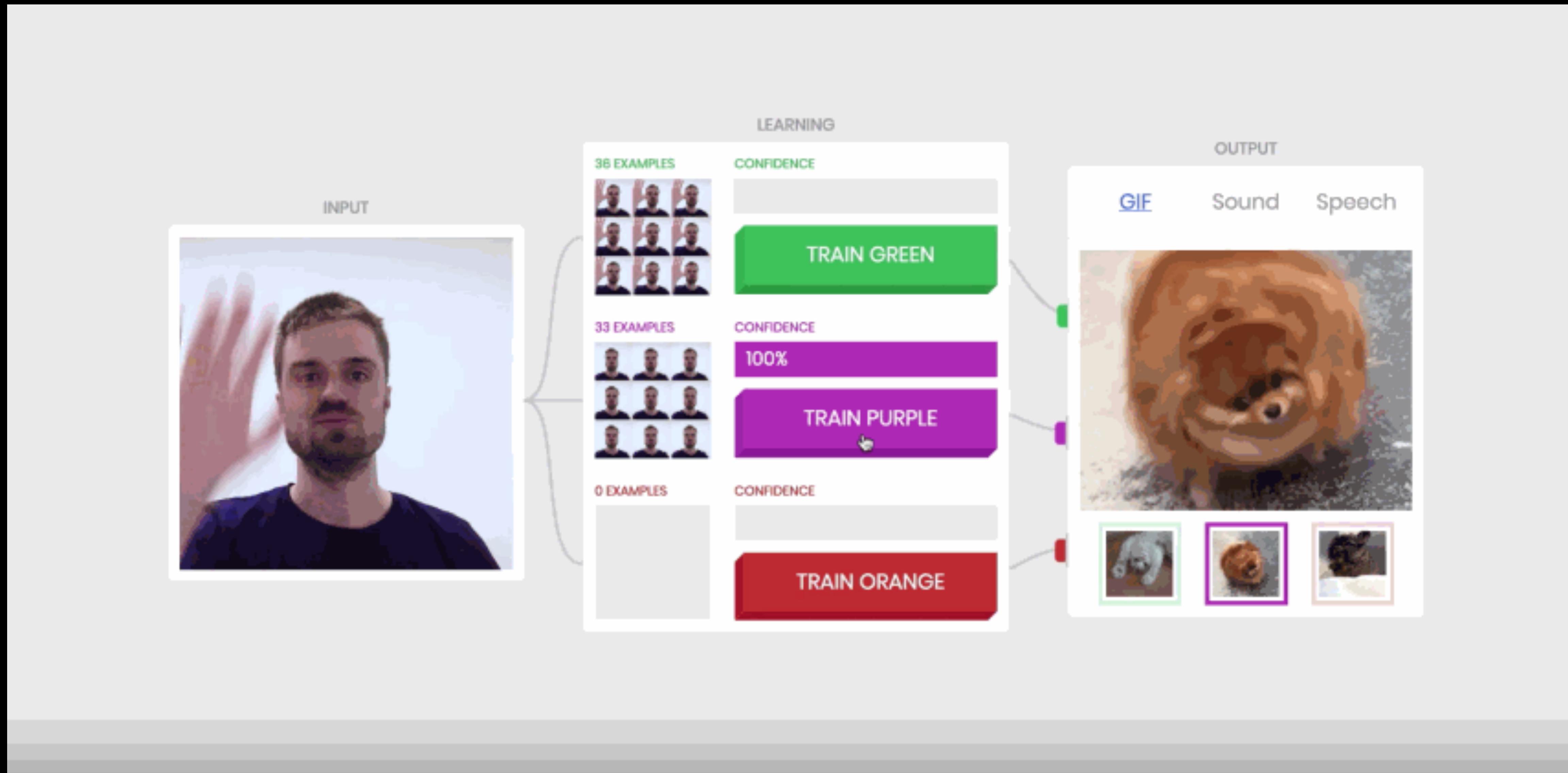
Fast Neural Style Transfer with Deeplearn.JS

Content  
face

Style  
Udnie, Francis Picabia

START STYLE TRANSFER

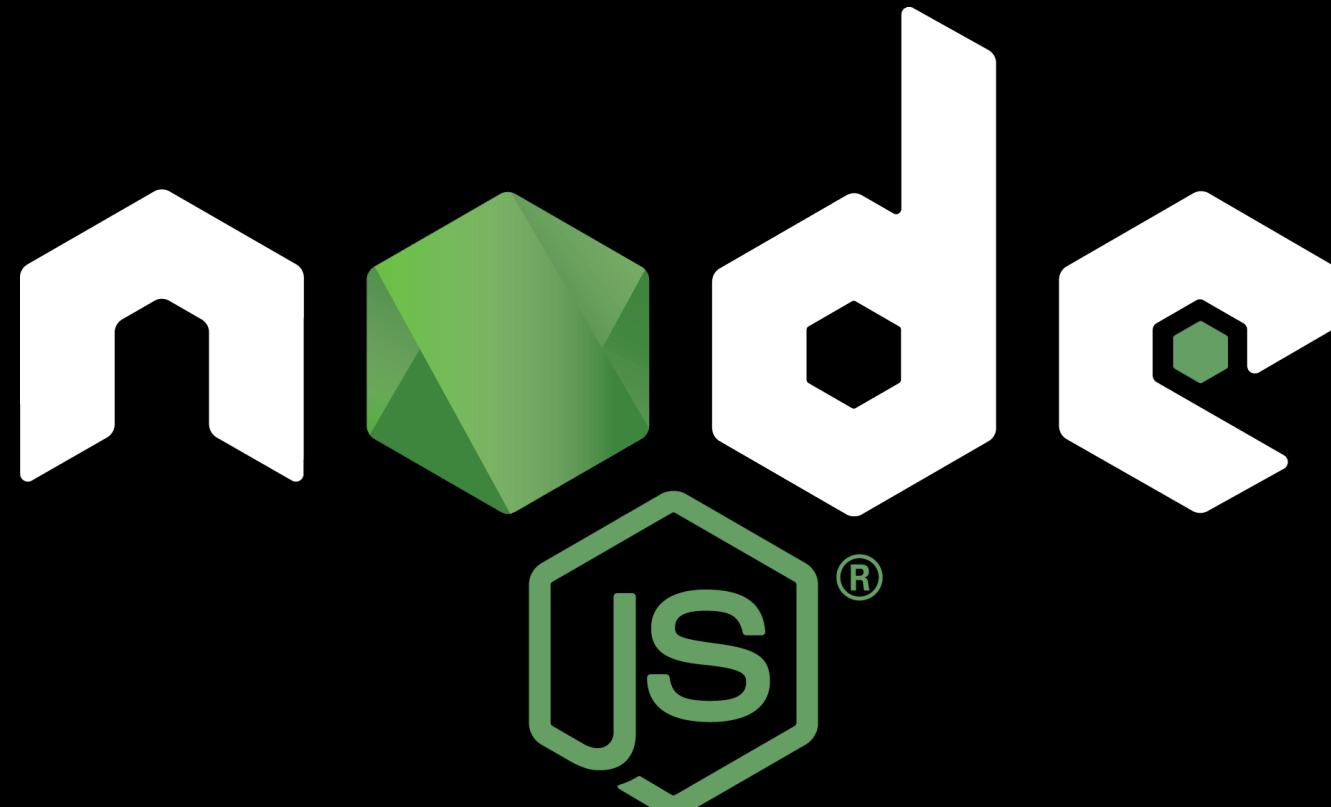
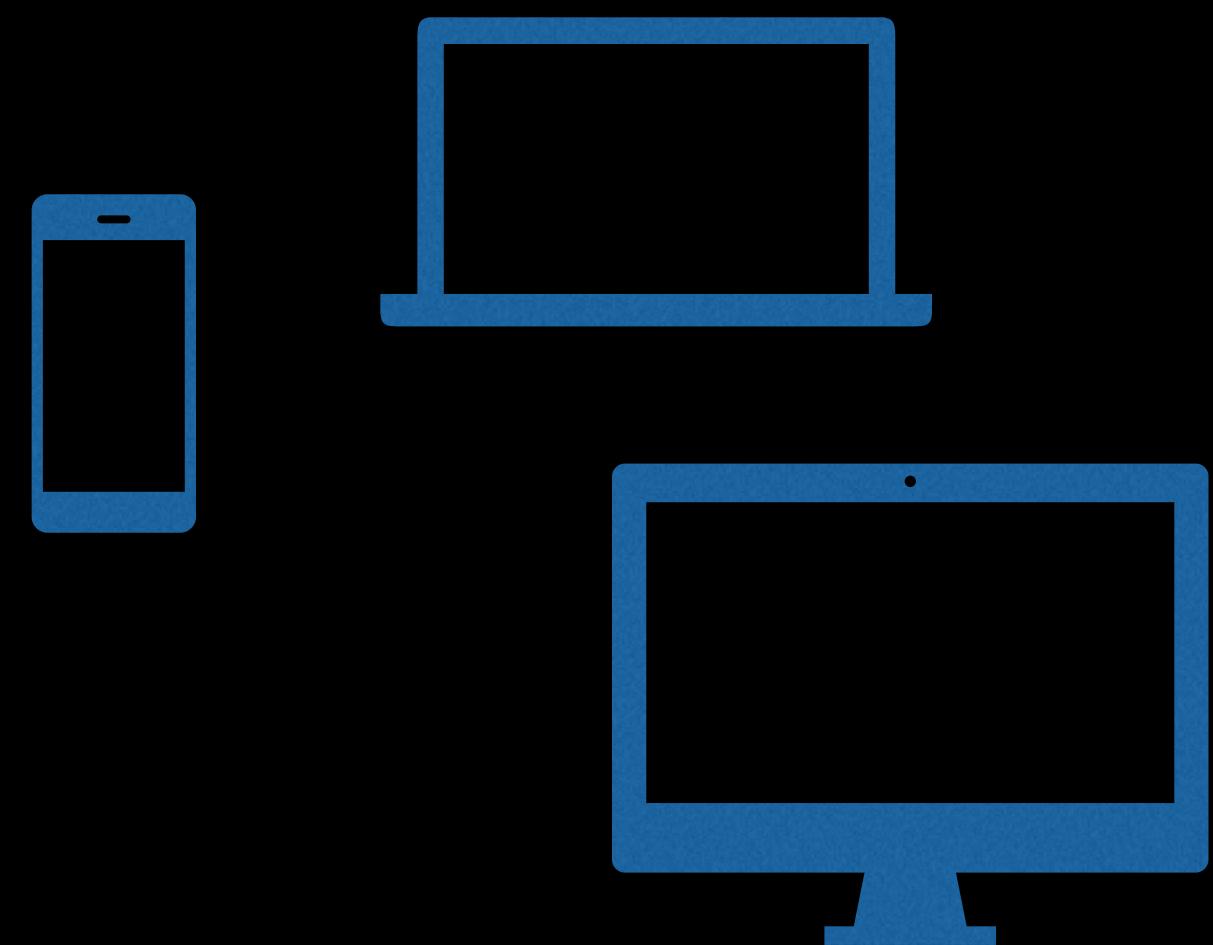
# deeplearn.js & Teachable Machine



# TensorFlow.js



X Drivers  
X Install



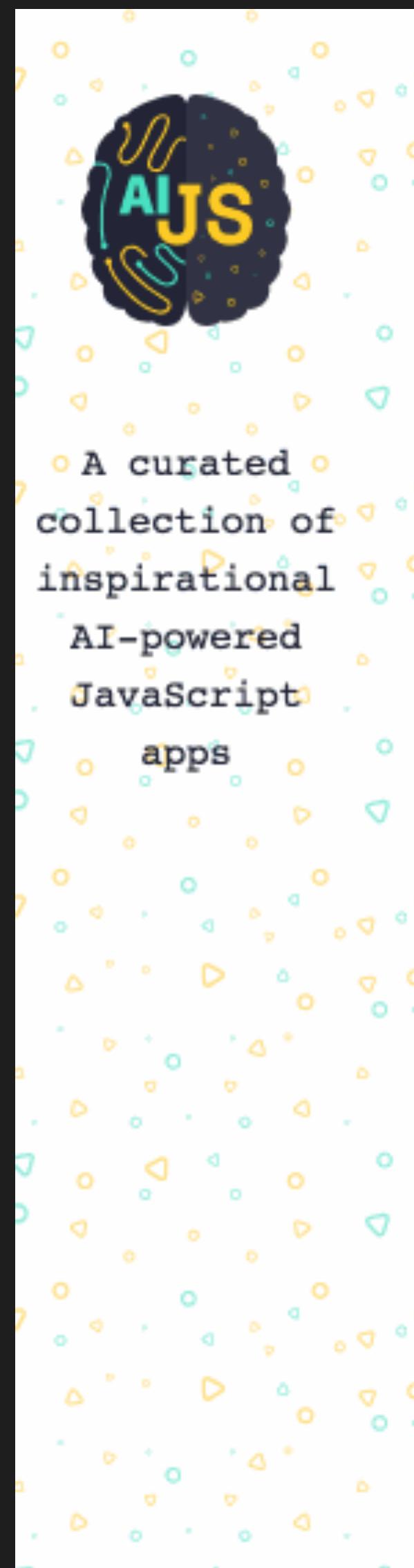
# TensorFlow.js

Create and Train Models

Load Pretrained Models

Retrain Existing Models

# aijs.rocks



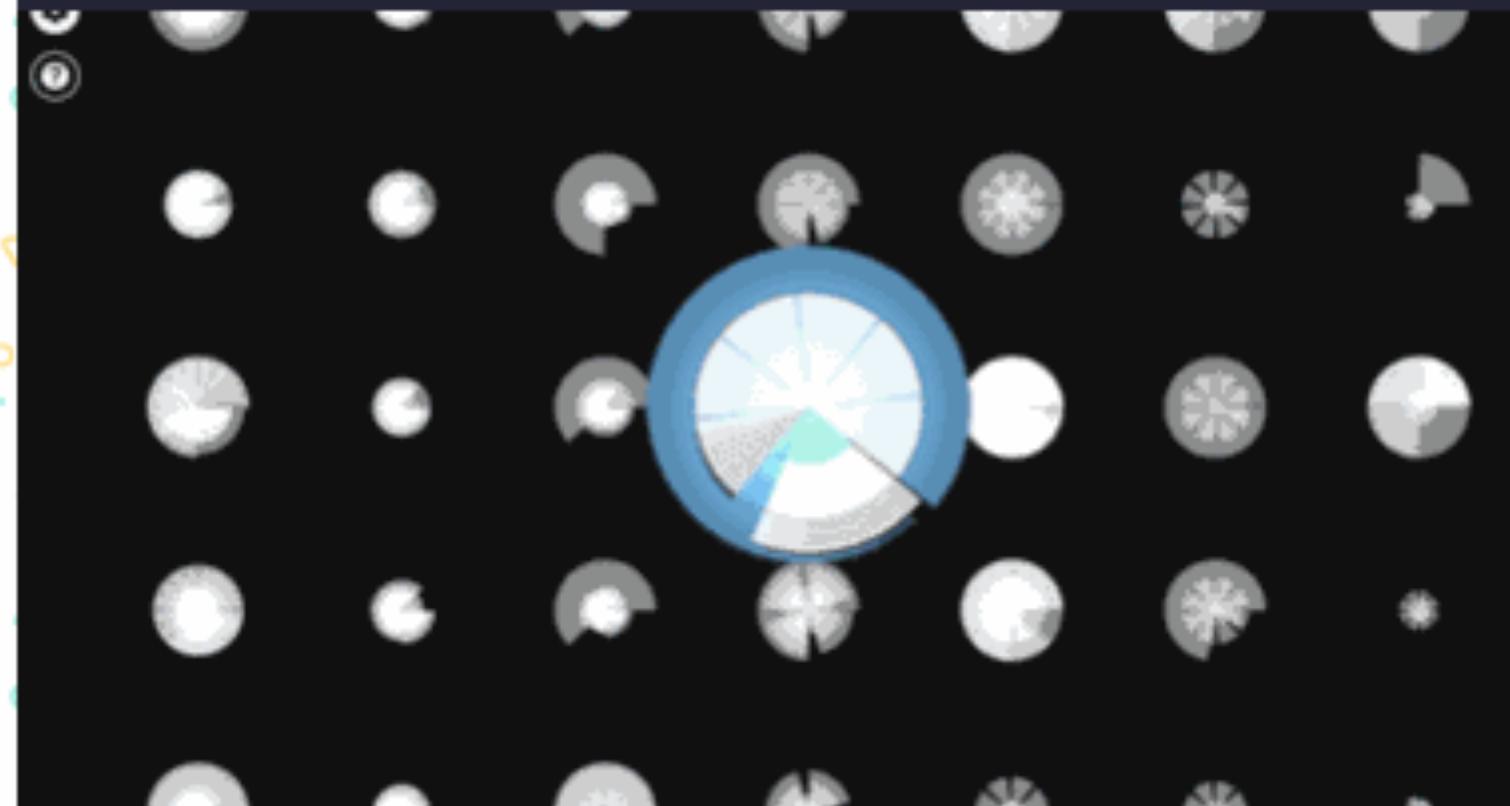
An interactive drum machine created with love for Music and AI.

Magenta.js Tone.js Magenta DrumRNN

 Gogul Ilango September 02, 2018

A curated collection of inspirational AI-powered JavaScript apps

## Musical Spinners From Latent Space

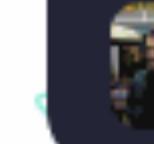


An AI-powered interactive music experience

TensorFlow.js Magenta.js Three.js

Automatically generated emoji captions for your images

Azure Computer Vision

 Alex Saunders August 30, 2018

## Image2Image



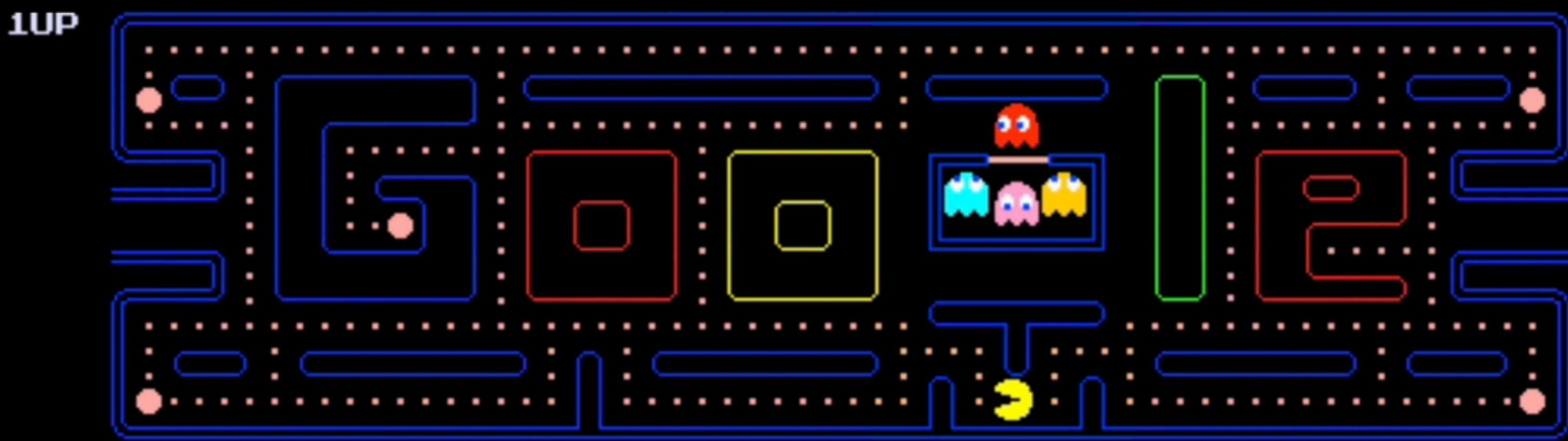
Make auto-generated images out of simple line drawings

TensorFlow.js Pix2Pix

About

Inspire

Submit



**TRAIN MODEL**      **PLAY**

Learning rate  
0.0001

Batch size  
0.4

Epochs  
20

Hidden units  
100

*Click to add the current camera view as an example for that control*

0 examples

0 examples

0 examples

0 examples

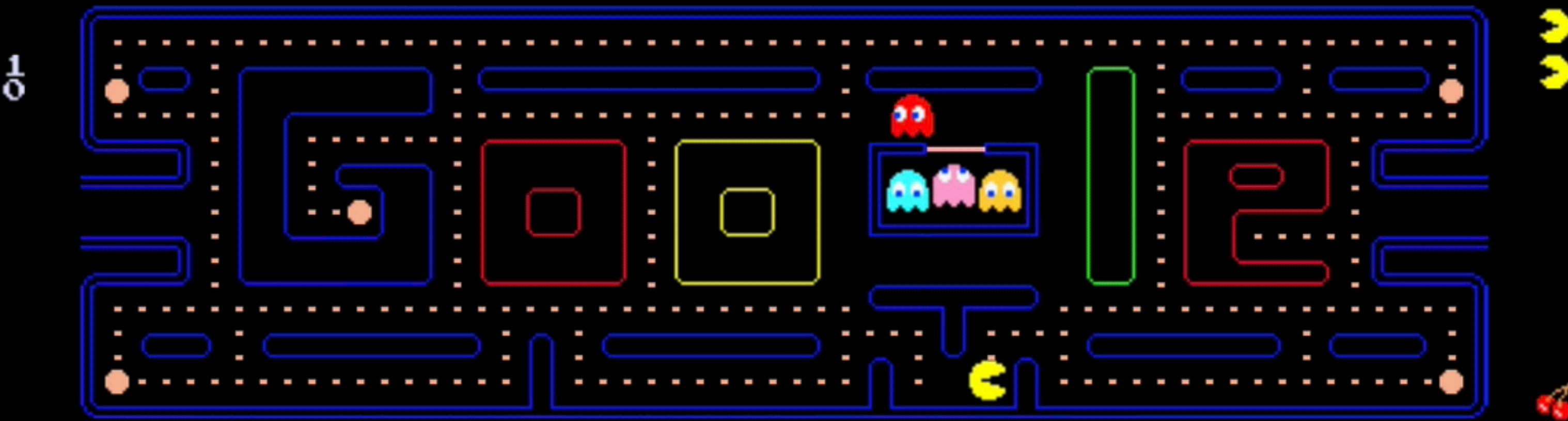
Add Sample

Add Sample

Add Sample

Ac\_Sample

The interface shows a camera feed of a person wearing a blue beanie with a cartoon face, positioned between the TRAIN MODEL and PLAY buttons. Below the camera feed are four control buttons: up, down, left, and right. Each button has an 'Add Sample' button below it. A callout text above the buttons says: "Click to add the current camera view as an example for that control". To the right of the buttons are four boxes labeled "0 examples" each, with "Add Sample" buttons at the bottom.



LOSS: 0.00000

PLAY

Click to add the current camera view as an example for that control

27 examples

26 examples

20 examples

20 examples

Learning rate  
0.0001

Batch size  
0.4

Epochs  
20

Hidden units  
100

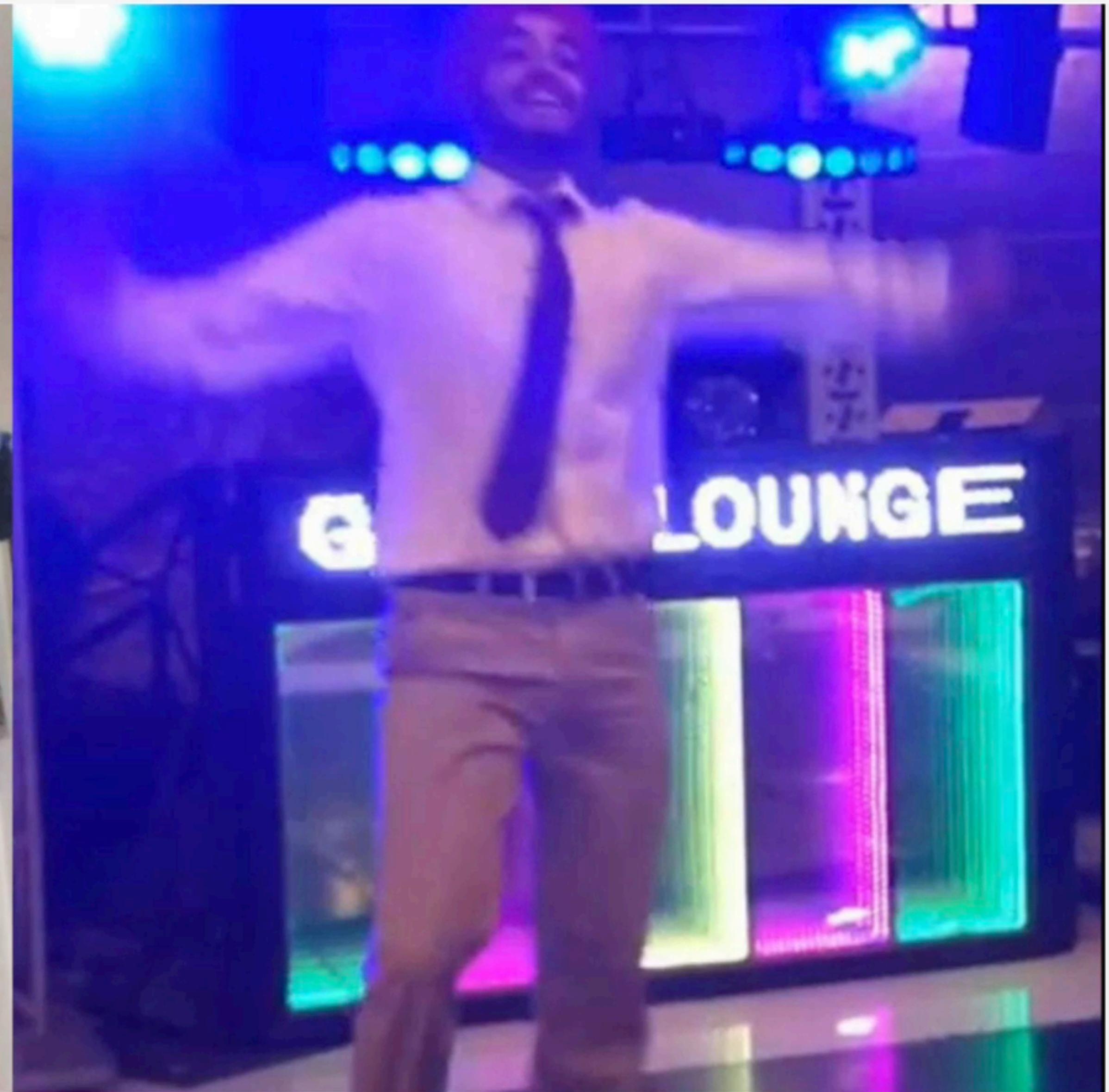
A central video feed shows a man with a beard and a blue beanie. To the left, a large red button labeled "PLAY" has a white hand cursor icon pointing to it. Below the video feed are five smaller video frames, each showing the same man. Each frame has a "Add Sample" button at the bottom. A dashed line connects the central "PLAY" button to the top frame. To the left of the video feed are four dropdown sliders: Learning rate (0.0001), Batch size (0.4), Epochs (20), and Hidden units (100). The top frame has a yellow border around its "Add Sample" button, while the others have black borders.

# MOVE MIRROR

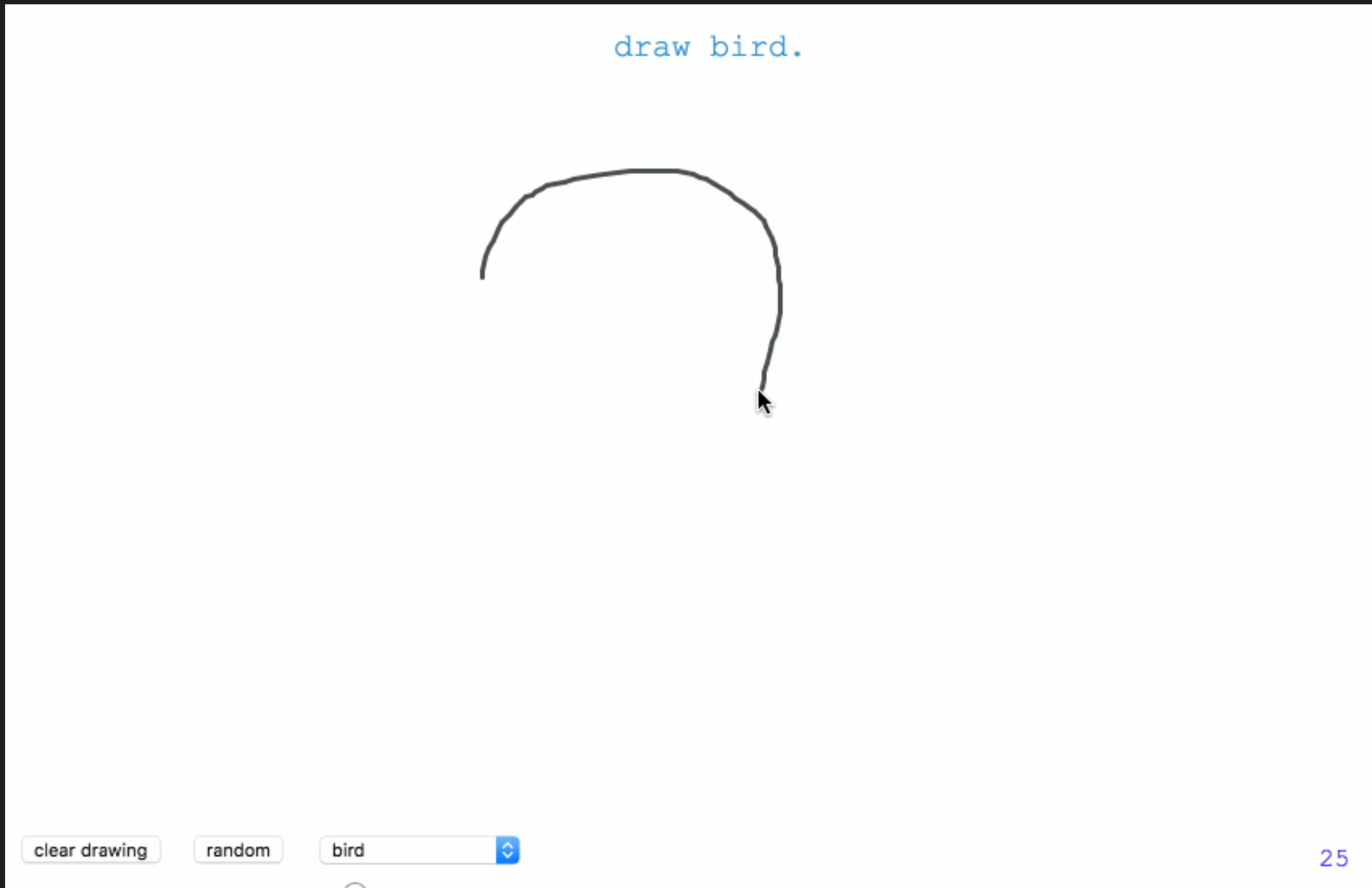


Make a GIF

# MOVE MIRROR



# Sketch RNN



# magenta.js



**magenta**

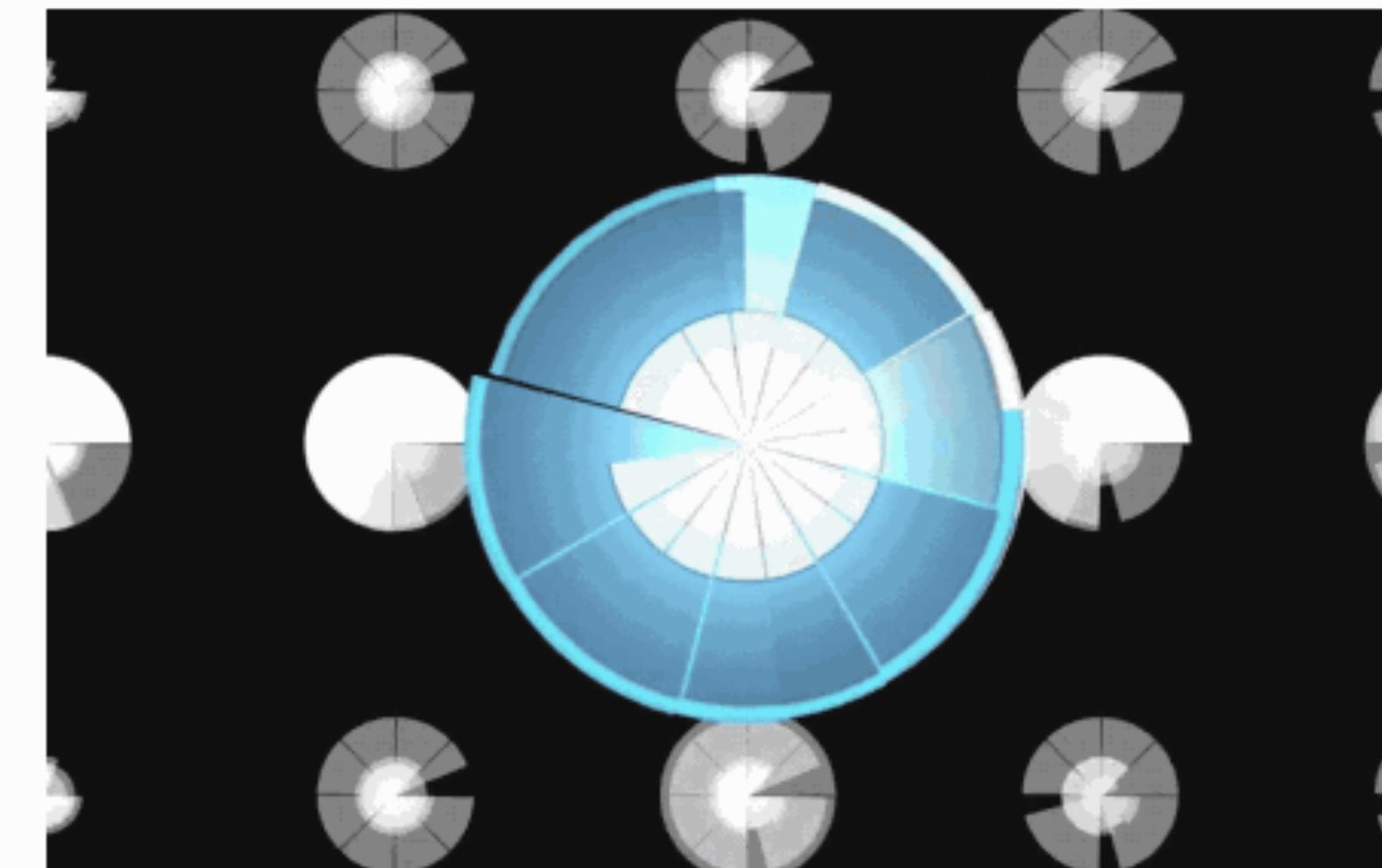
[Home](#) [Blog](#) [Demos](#) [Publications](#) [Datasets](#) [Discuss](#) [GitHub](#)

---

## Featured

The Incredible Musical Spinners From Latent Space [ [play](#) | [blog](#) | [code](#) ]

Tero Parviainen ( teropa , teropa )



An “AI-powered interactive music experience” that presents a playable 7x7 grid of musical measures. Each row is an interpolation between two endpoints and each column is the “same” measure interpreted in 7 different chords, e.g. C Major. You can choose the key to play in.

# TensorFlow.js



```
import * as tf from '@tensorflow/tfjs';
```

# TensorFlow.js



```
<html>
  <head>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@0.10.0"> </script>

    <script>
      const x = tf.tensor([1,2,3]);
      tf.print(x);    // Tensor [1, 2, 3]
    </script>
  </head>
</html>
```

# TensorFlow.js

Core API 

Layers API 

# TensorFlow.js



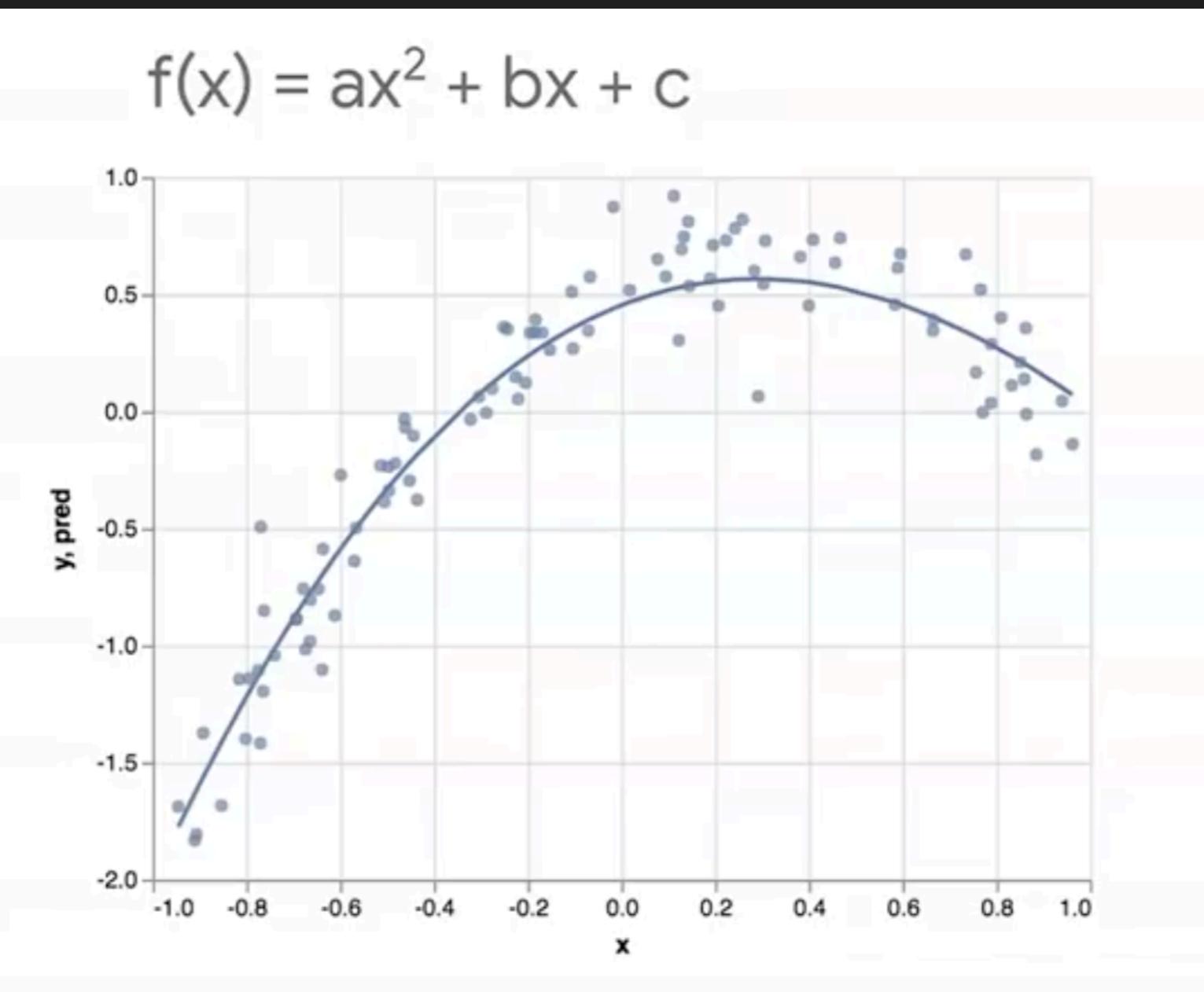
```
const shape = [2, 3]; // 2 rows, 3 columns
const a = tf.tensor([1.0, 2.0, 3.0, 10.0, 20.0, 30.0], shape);
a.print();
// Output: [[1 , 2 , 3 ],
//           [10, 20, 30]]

// The shape can also be inferred:
const b = tf.tensor([[1.0, 2.0, 3.0], [10.0, 20.0, 30.0]]);
b.print();
// Output: [[1 , 2 , 3 ],
//           [10, 20, 30]]
```

# Core API



## Polynomial Regression



## TensorFlow.js



```
// Define function
function predict(input) {
  // y = a * x ^ 2 + b * x + c
  // More on tf.tidy in the next section
  return tf.tidy(() => {
    const x = tf.scalar(input);

    const ax2 = a.mul(x.square());
    const bx = b.mul(x);
    const y = ax2.add(bx).add(c);

    return y;
  });

  // Define constants: y = 2x^2 + 4x + 8
  const a = tf.scalar(2);
  const b = tf.scalar(4);
  const c = tf.scalar(8);

  // Predict output for input of 2
  const result = predict(2);
  result.print() // Output: 24
```

# Layers API



## TensorFlow.js



```
const model = tf.sequential();
model.add (
  tf.layers.dense({
    units: 20,
    activation: 'relu',
  }),
);

const optimizer = tf.train.sgd(LEARNING_RATE);
model.compile({optimizer, loss: 'meanSquaredError'});
model.fit({x: data, y: labels});
```

# TensorFlow.js

## Memory Management



```
// tf.tidy takes a function to tidy up after
const average = tf.tidy(() => {
  // tf.tidy will clean up all the GPU memory used by tensors inside
  // this function, other than the tensor that is returned.
  //
  // Even in a short sequence of operations like the one below, a number
  // of intermediate tensors get created. So it is a good practice to
  // put your math ops in a tidy!
  const y = tf.tensor1d([1.0, 2.0, 3.0, 4.0]);
  const z = tf.ones([4]);

  return y.sub(z).square().mean();
});

average.print() // Output: 3.5
```

# TensorFlow.js

## Memory Management



```
const x = tf.tensor2d([[0.0, 2.0], [4.0, 6.0]]);  
const x_squared = x.square();  
  
x.dispose();  
x_squared.dispose();
```

# TensorFlow.js

## Load pre-trained models



```
const mobilenet = await tf.loadModel(  
  'https://storage.googleapis.com/tfjs-models/tfjs/mobilenet_v1_0.25_224/model.json');
```

# TensorFlow.js

## Transfer Learning



```
async function loadMobilenet() {
  const mobilenet = await tf.loadModel(
    'https://storage.googleapis.com/tfjs-models/tfjs/mobilenet_v1_0.25_224/model.json');

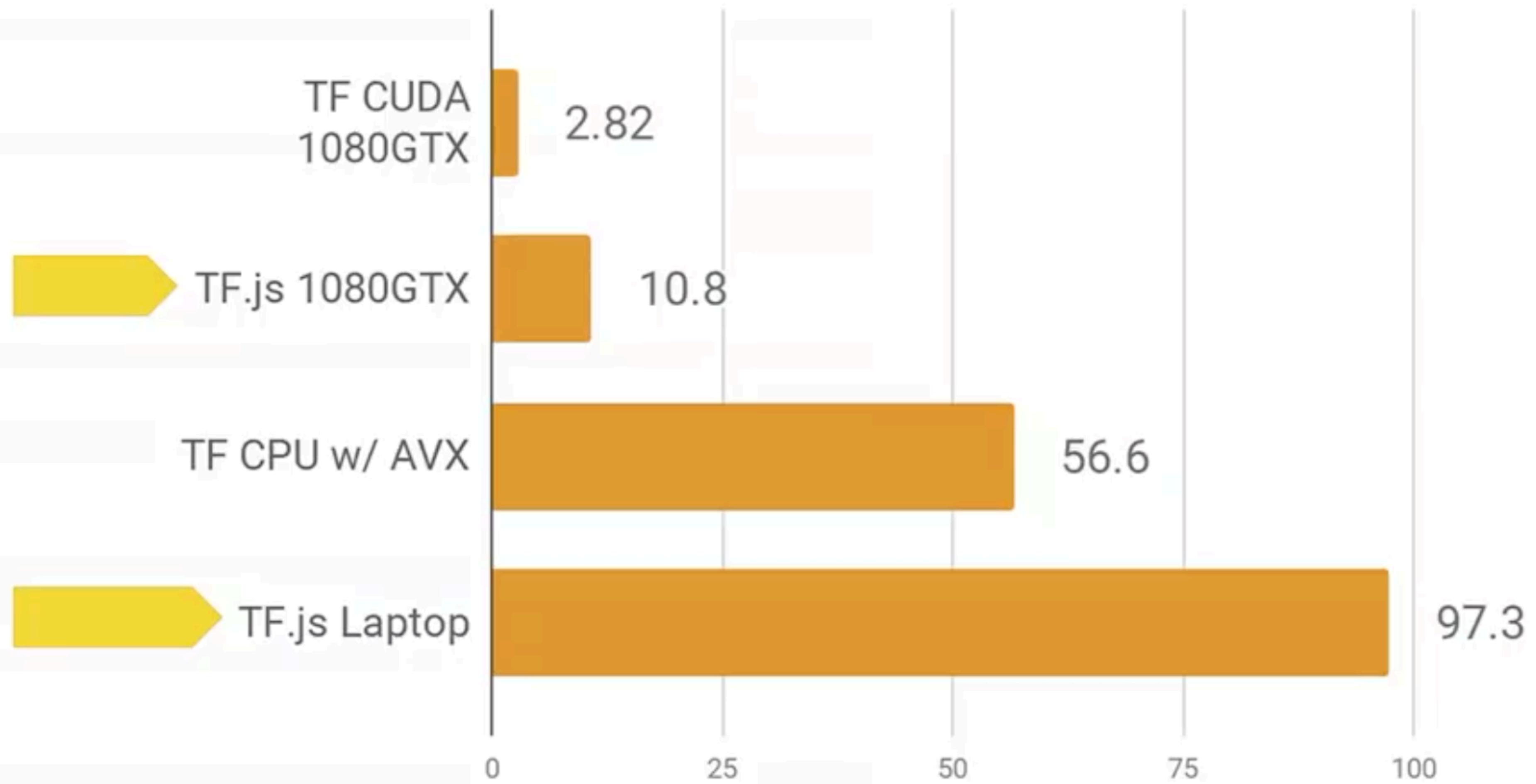
  // Return a model that outputs an internal activation.
  const layer = mobilenet.getLayer('conv_pw_13_relu');
  return tf.model({inputs: model.inputs, outputs: layer.output});
};
```

# TensorFlow.js

Create and Train Models

Load Pretrained Models

Retrain Existing Models



Inference Time (ms) of MobileNet 1.0\_224

Average of 200 runs

# Applications of

## ML with

### JavaScript

# Getting Started

# Getting Started & Resources

[js.tensorflow.org](https://js.tensorflow.org)

[github.com/tensorflow/tfjs](https://github.com/tensorflow/tfjs)

[github.com/tensorflow/tfjs-examples](https://github.com/tensorflow/tfjs-examples)

[aijs.rocks](https://aijs.rocks)

[ml5.js](https://ml5.js)

The Coding Train

# The Future of ML & JavaScript