Artificial Neural Networks

INTRODUCTION

DECIDING

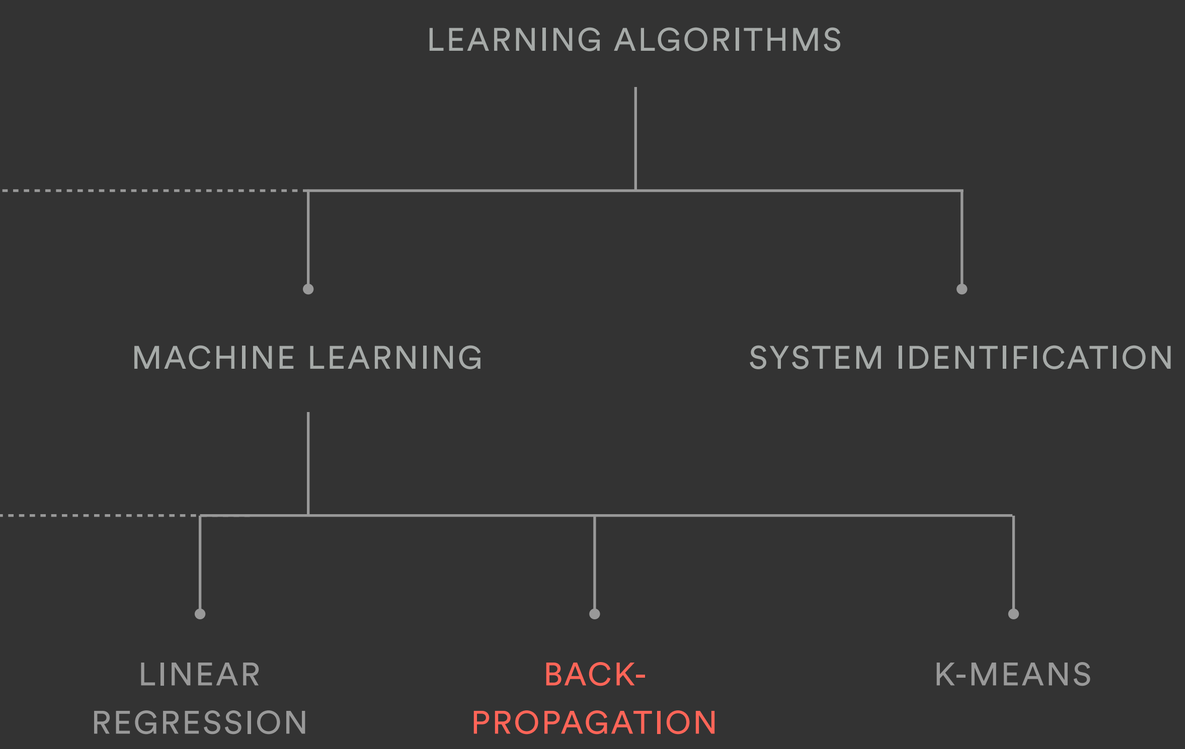
LEARNING

CORRECTNESS

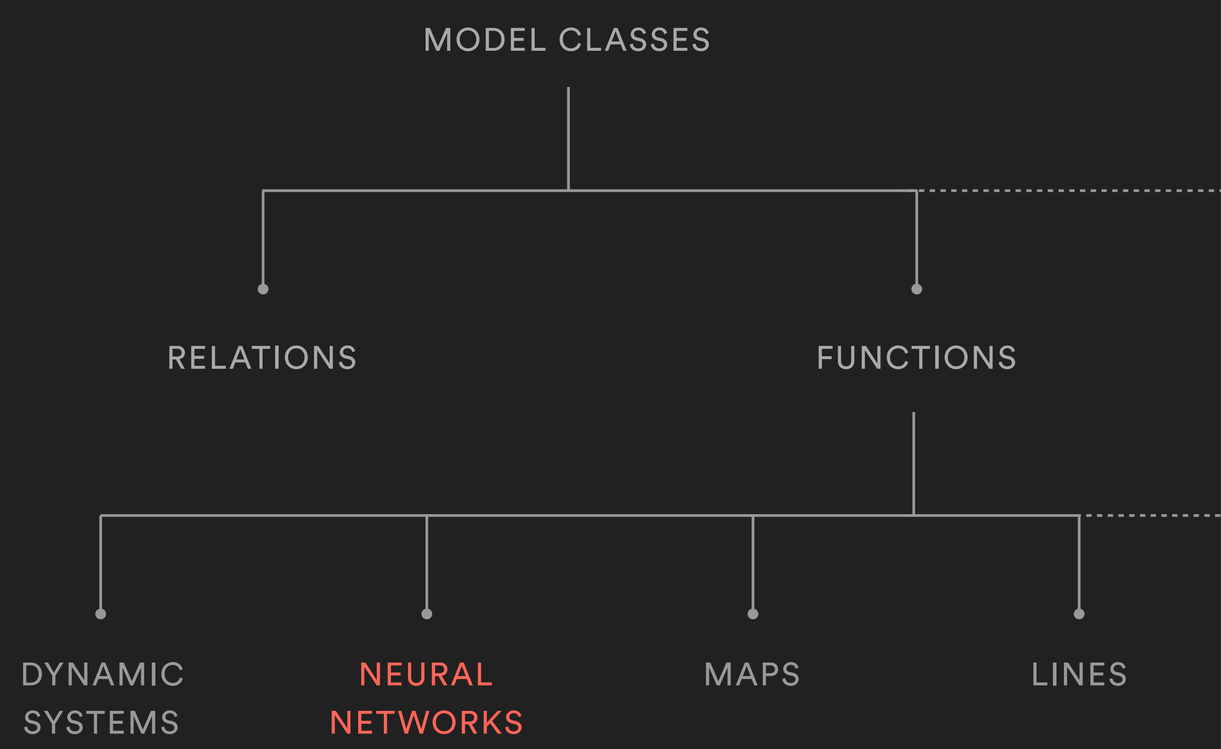
Learning

Model

Learning



Model



What is an Artificial Neural Network (ANN)?

What is an Artificial Neural Network (ANN)?

An ANN is a universal function approximator

Deep Learning

AN ANN WITH MORE NEURONS AND MORE LAYERS

Problems solved with Neural Networks

CLASSIFICATION

IMAGES RECOGNITION

Handwritten letters
Faces

AUDIO RECOGNITION

Speech

PATTERN RECOGNITION

Spam

CLUSTERING

GROUPING

Customers by similar characteristics

Geographic distances together for deliveries

REGRESSION

PREDICTION

Stock picking

income based based on location

airline passengers based on the time of year

pinterest.com

Visually similar results

jacket | bombers jackets | bomber | john varvatos

Free shipping and returns on Burberry Brit 'Irwin'...

Theory Leather Bomber Jacket

Rick Owens Leather Bomber Jacket

Collar Parka

Reebok

Mower Leather Bomber Jacket

BOSS Black 'Cosey' Trim Fit Jacket available at... #Nordstrom

Pistel Whip Striped Hoodie

cardiogr.am

CS2 | CS | ConvNetJS d... | 12a: Neural... | 3Blue1Brown... | Thinking vis... | tensorflow c... | neural nets e... | Frank Chen –... | The | ca | Cardiogram...

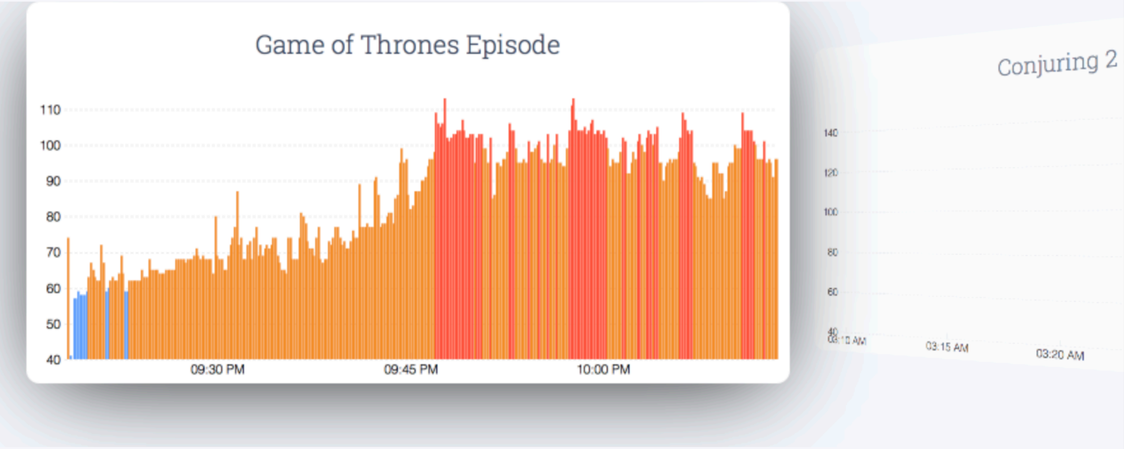
cardiogram

About Research For employers Blog

What's your heart telling you?

Your heart beats 102,000 times per day, and it reacts to everything that happens in your life —what you're eating, how you exercise, a stressful moment, or a happy memory.

Download on the App Store GET IT ON Google Play



 Privacy Terms of Use Support Contact us

THE NEW YORKER

D.I.Y. ARTIFICIAL INTELLIGENCE COMES TO A JAPANESE FAMILY FARM

By Amos Zeeberg August 10, 2017

For decades, Makoto Koike's mother has been sorting cucumbers by hand. Now he is trying to teach a machine to replace her.

Photograph by Yagi Studio / Getty

Not much about Makoto Koike's adult life suggests that he would be a farmer. Trained as an engineer, he spent most of his career in a busy urban section of Aichi Prefecture, Japan, near the headquarters of the Toyota Motor Corporation, writing software to control cars. Koike's


宅急便
L,2L
B□□


宅急便
L,2L
B□□

ANN Implementation Overview

MODEL

STEP 1

Define the input

STEP 2

Define the topology of the neural net
(i.e., the layers of neurons and their connections)

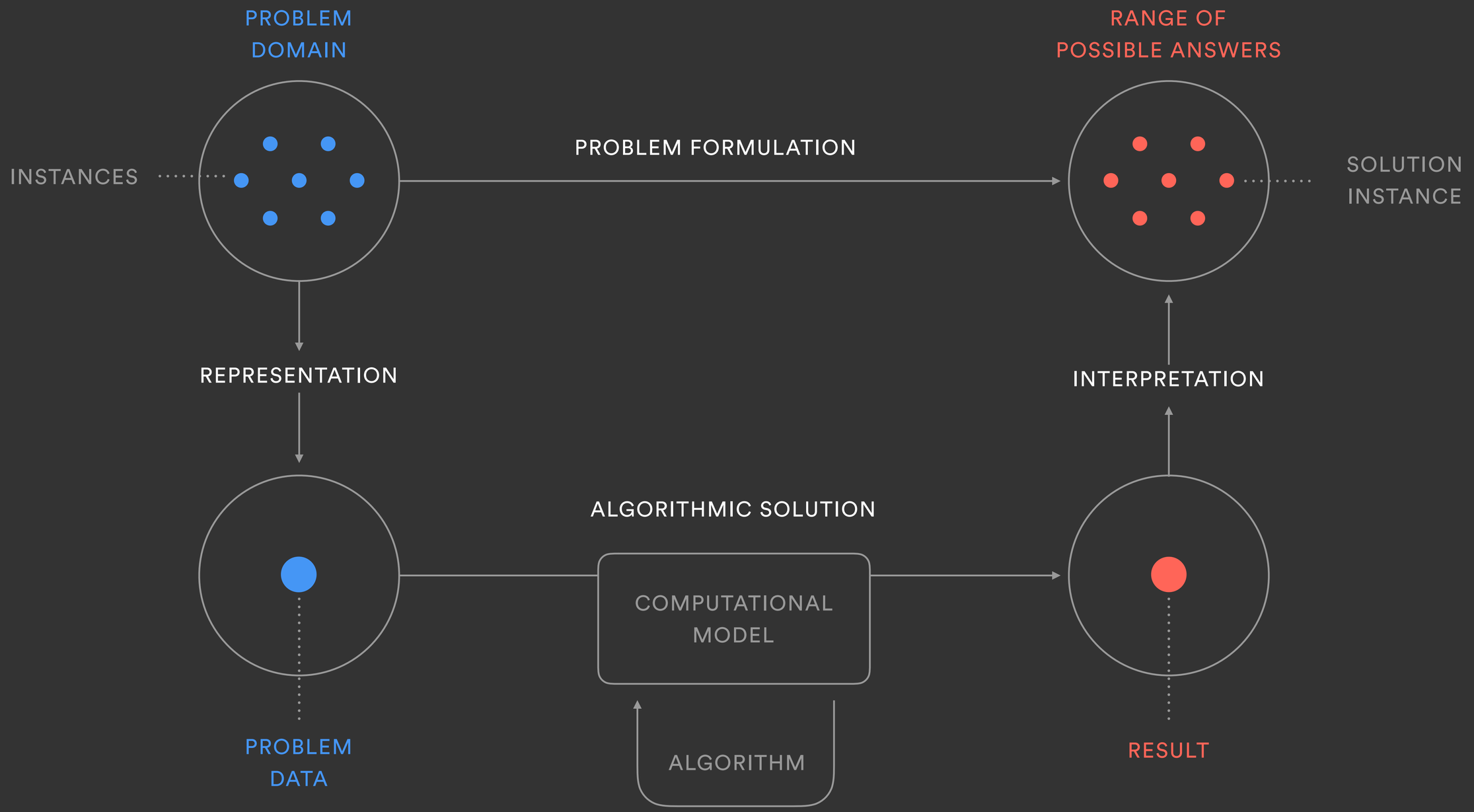
TRAINING THE MODEL

STEP 3

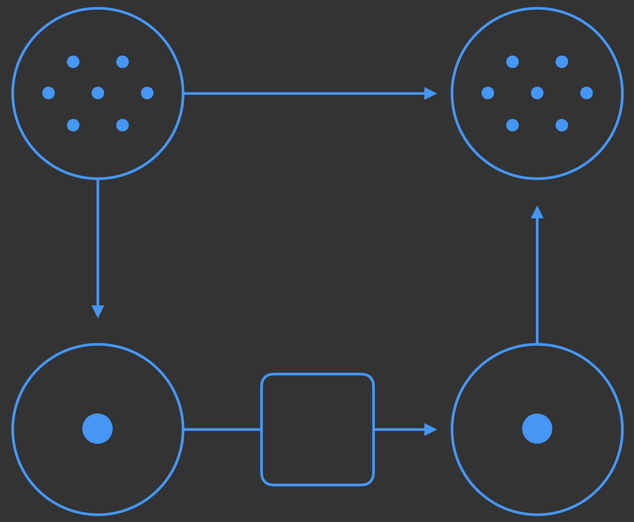
Train the neural net on examples of the problem

STEP 4

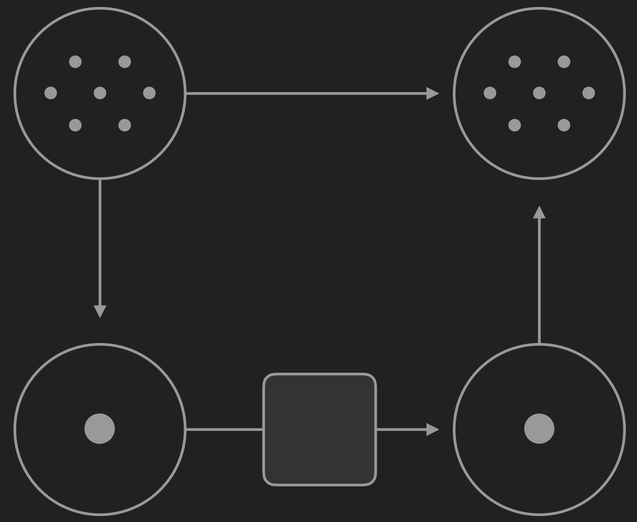
Run the trained neural net to solve new examples of the problem


PROBLEM
DOMAIN
RANGE OF
POSSIBLE ANSWERS
INSTANCES
PROBLEM FORMULATION
SOLUTION
INSTANCE
REPRESENTATION
INTERPRETATION
ALGORITHMIC SOLUTION
COMPUTATIONAL
MODEL
PROBLEM
DATA
ALGORITHM
RESULT

Learning

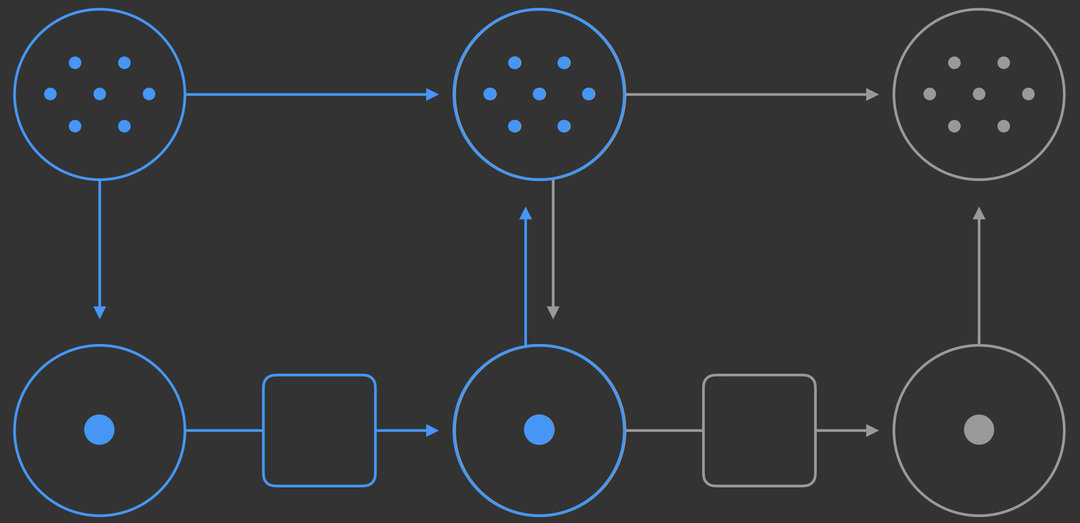


Deciding

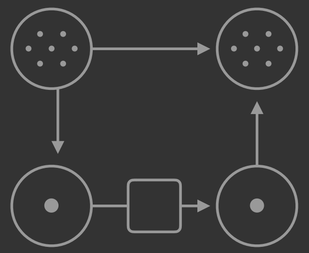


Learning

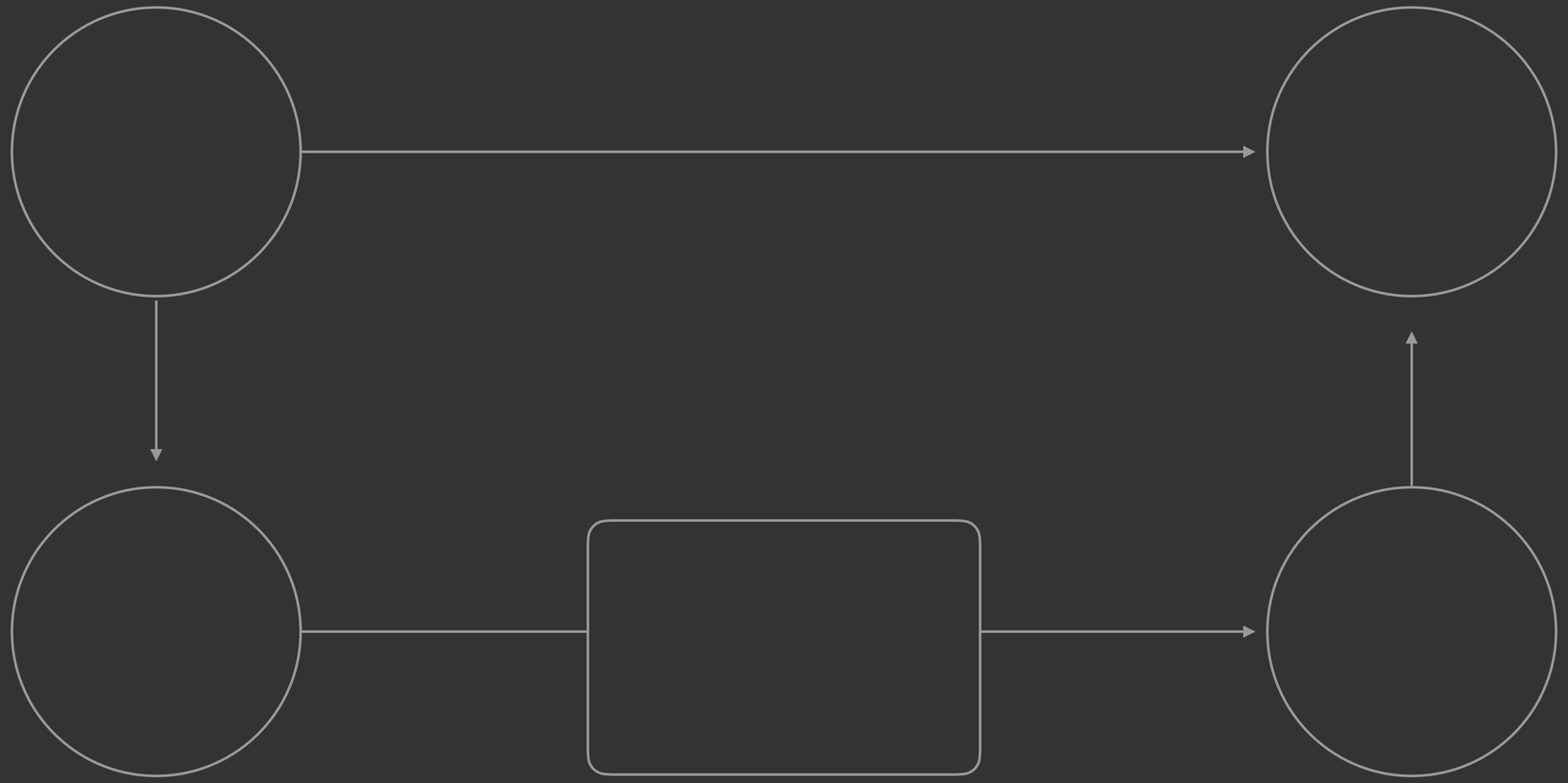
Deciding



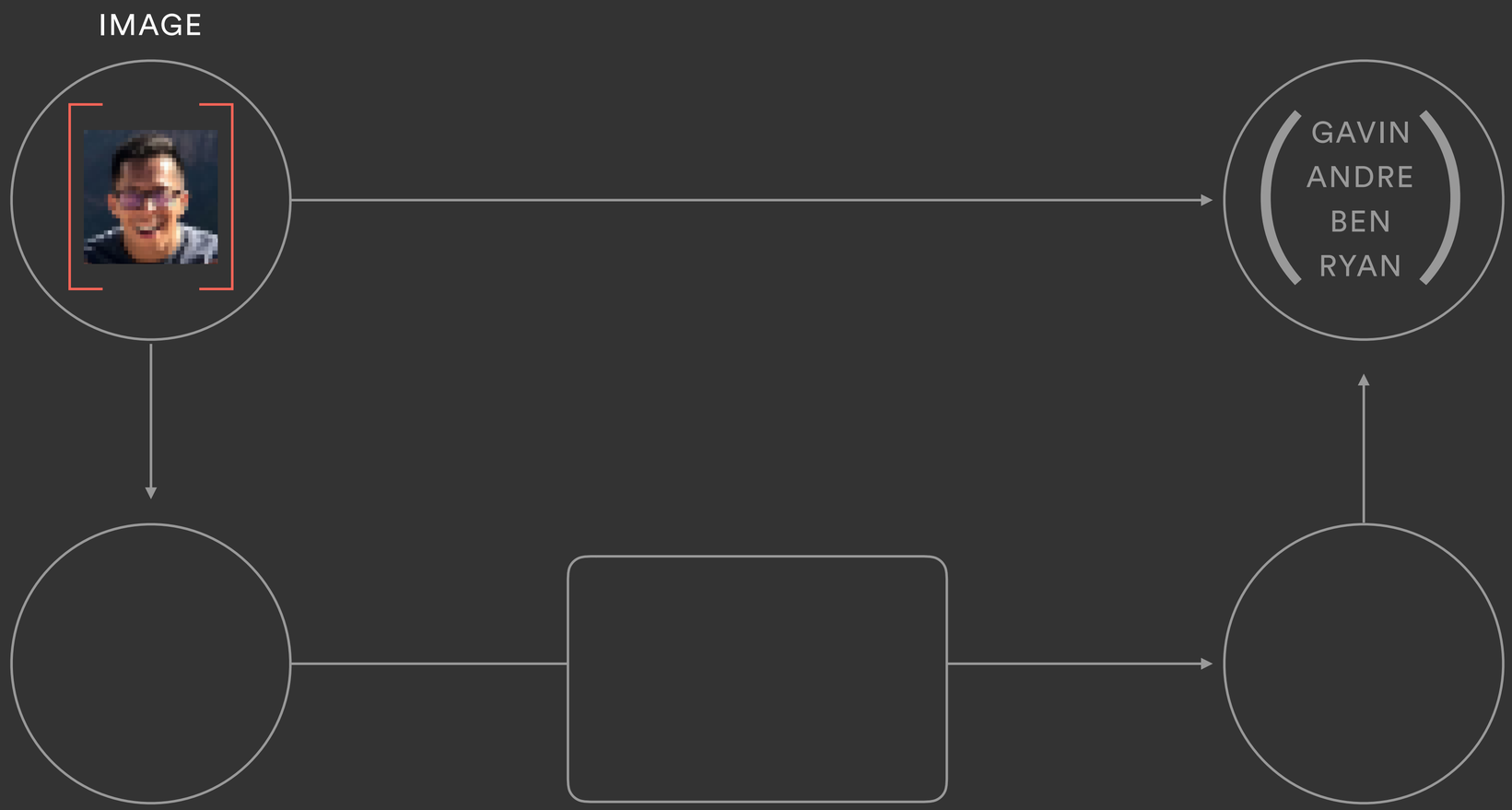
INTRODUCTION | DECIDING | LEARNING | CORRECTNESS



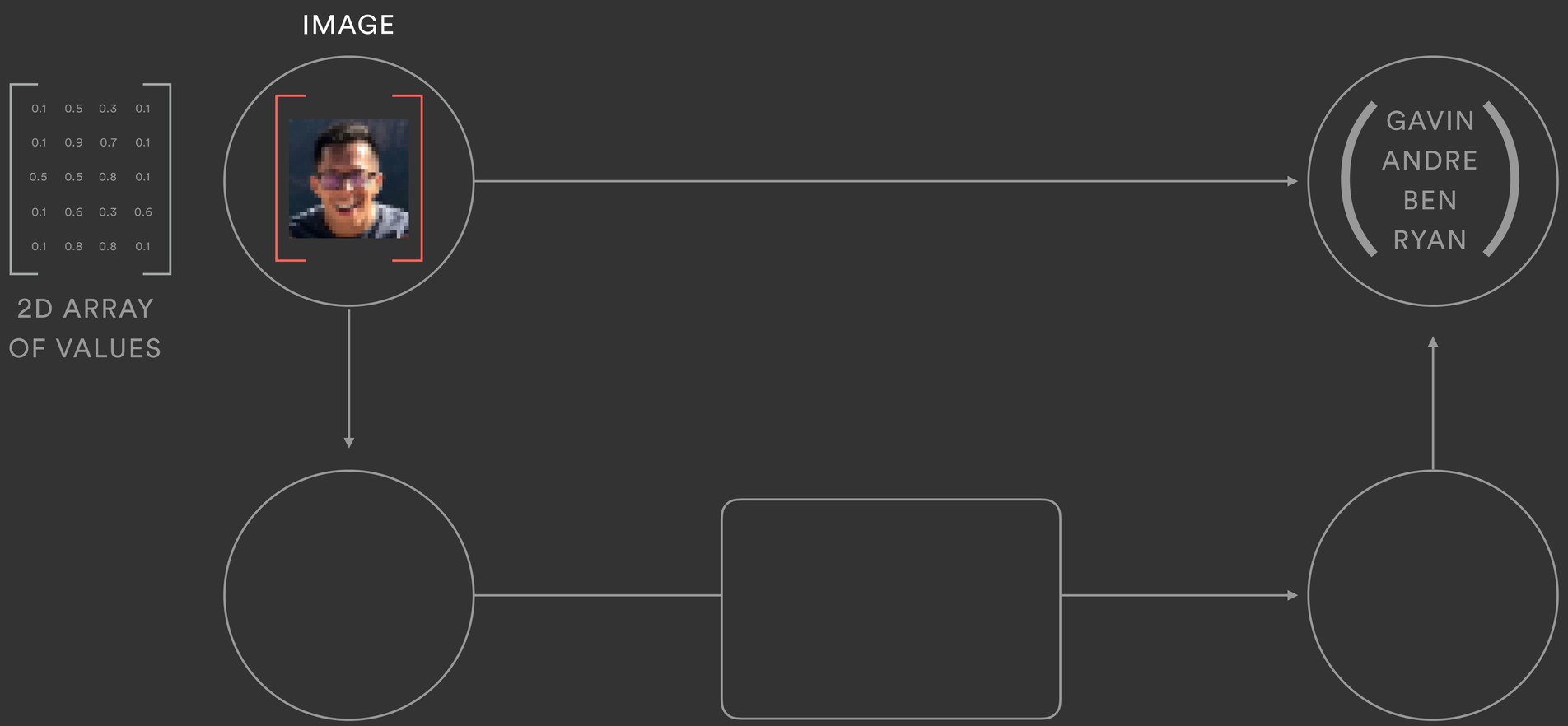
Decision Making



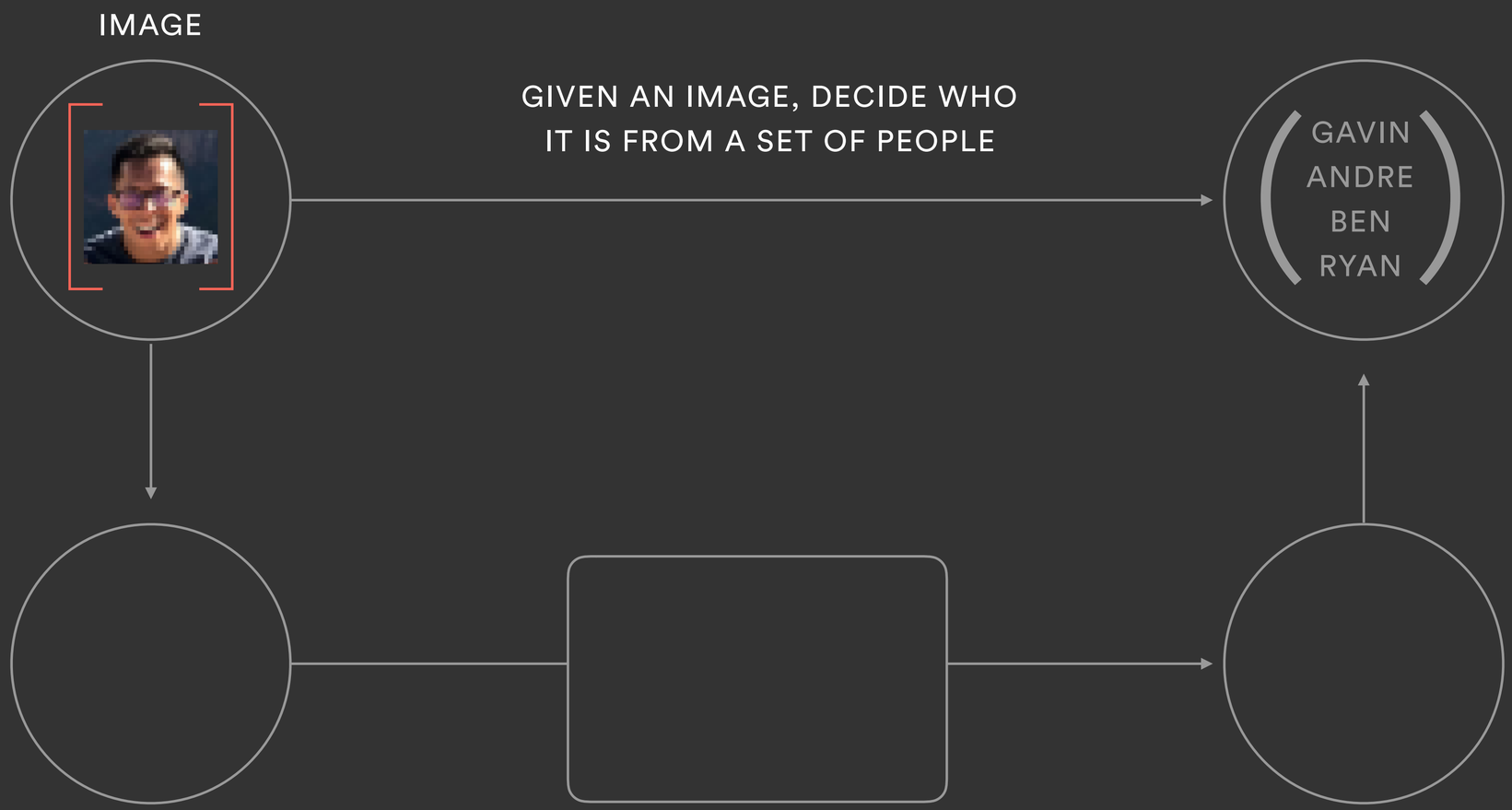
Decision Making



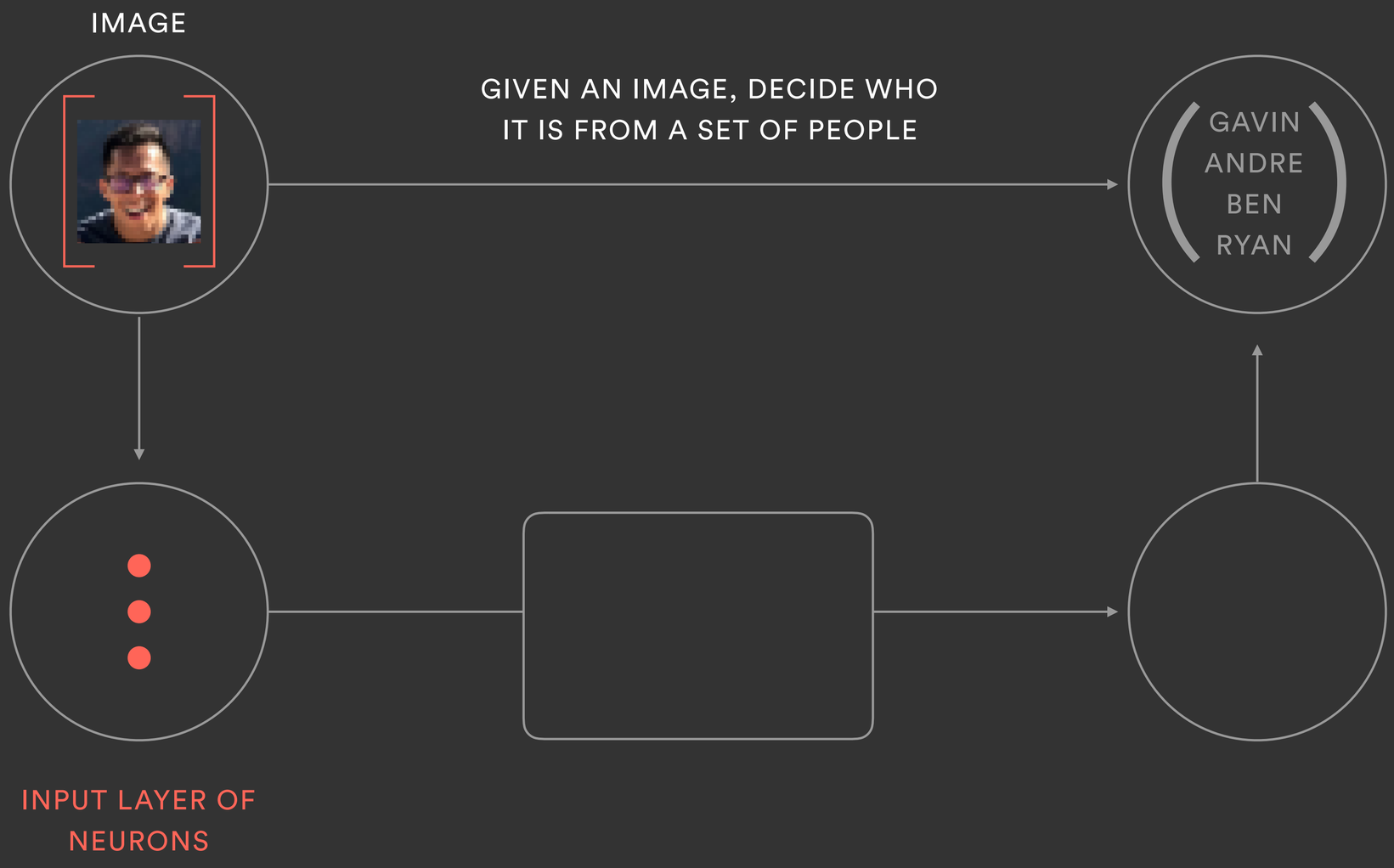
Decision Making



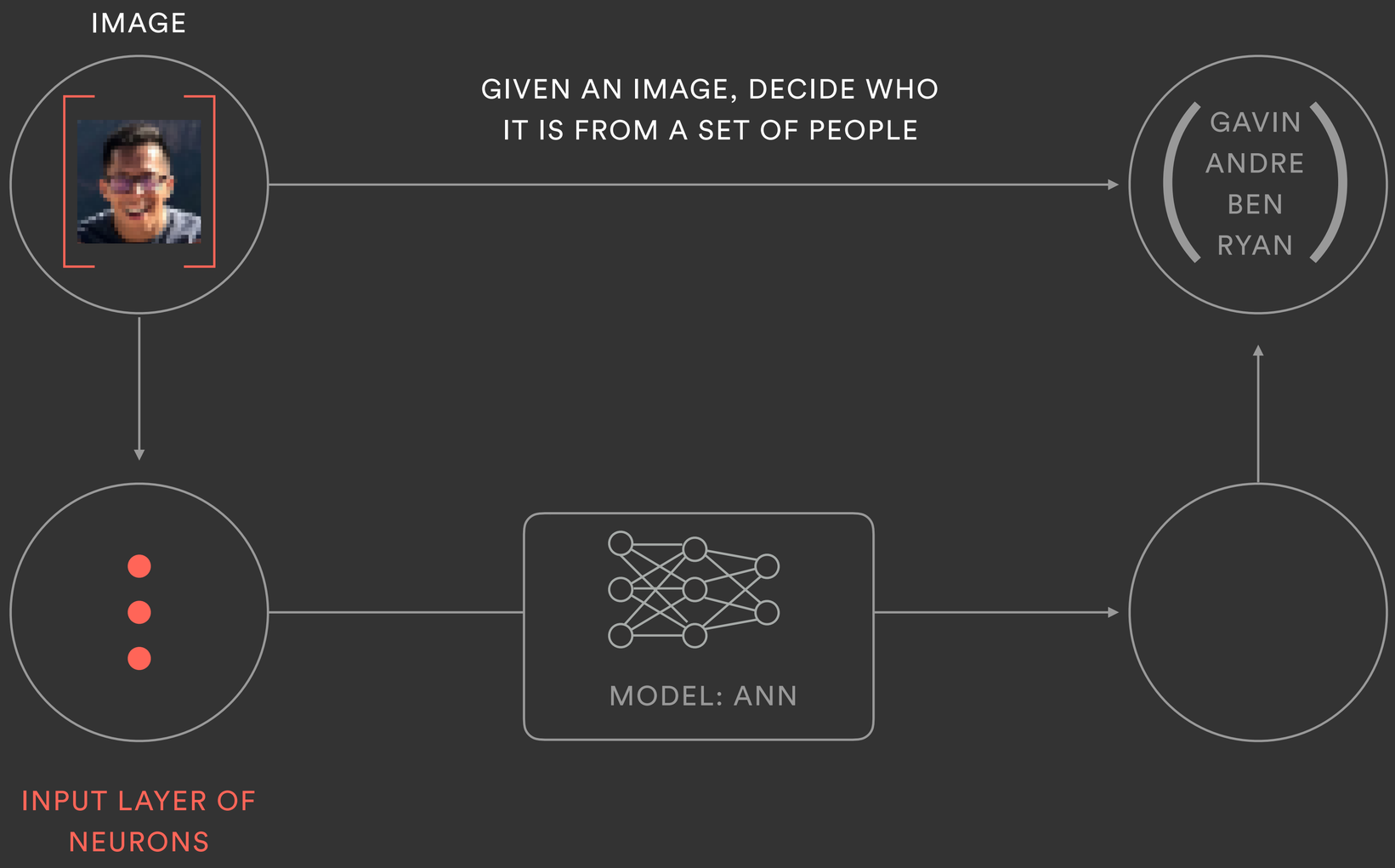
Decision Making



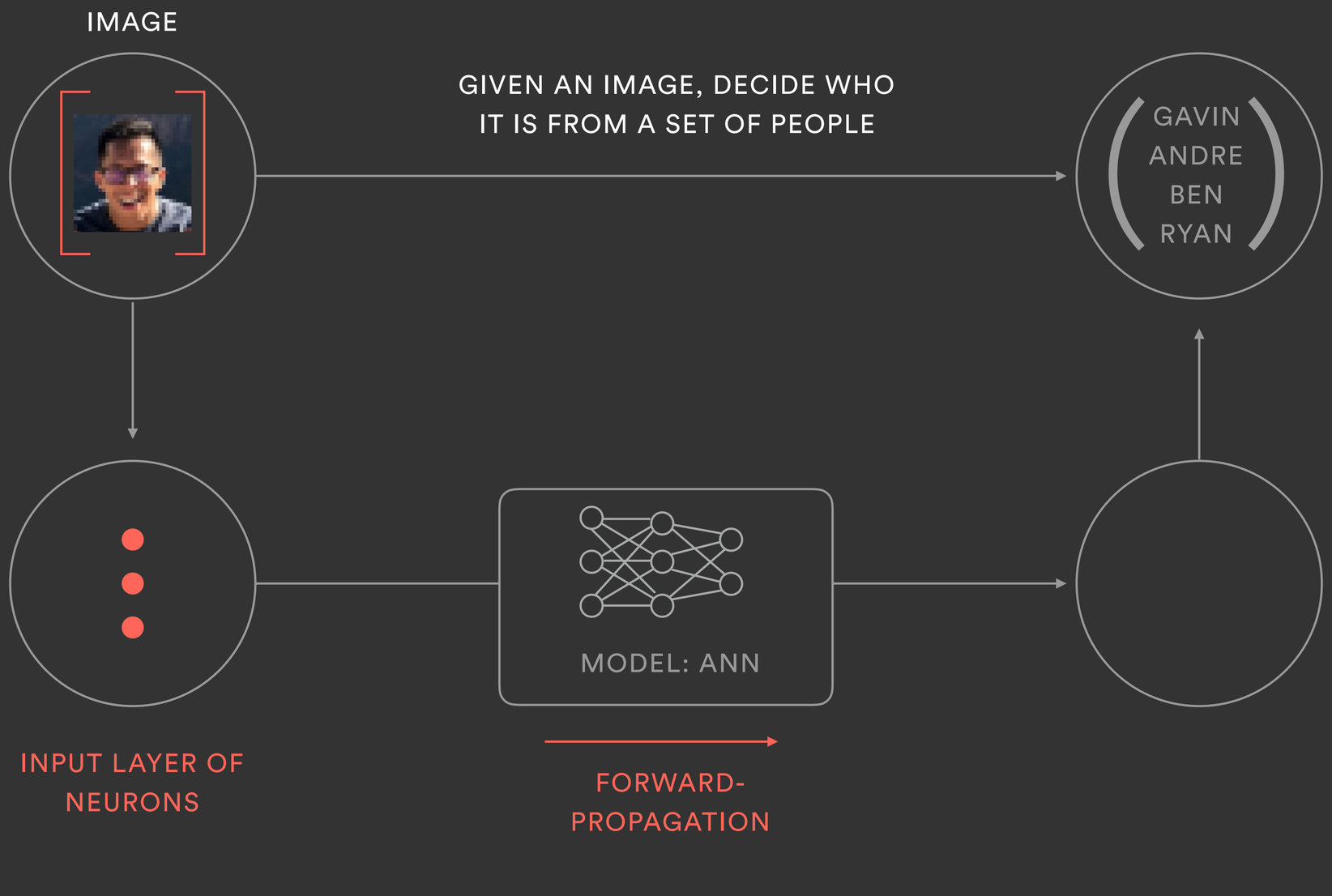
Decision Making



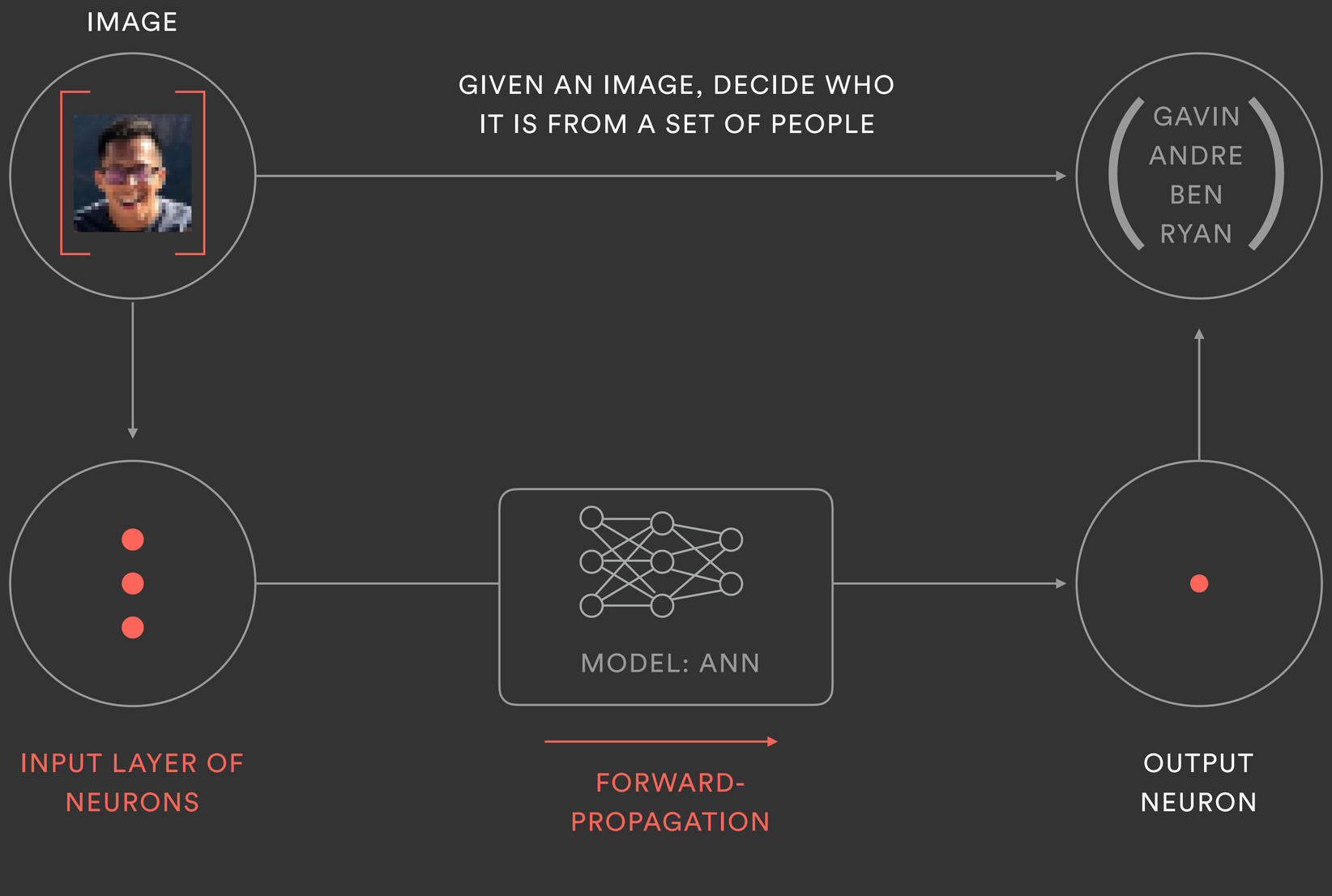
Decision Making



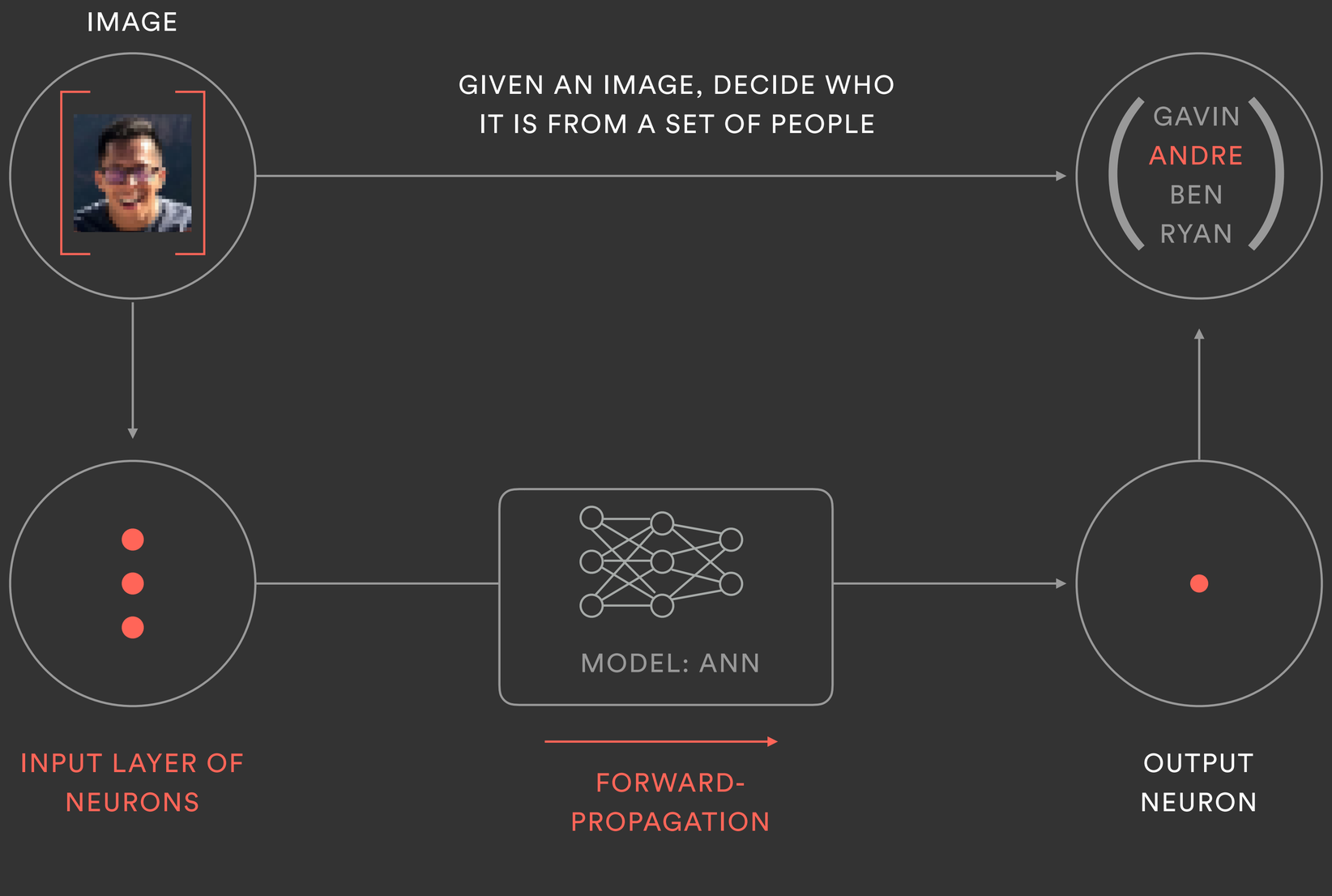
Decision Making



Decision Making



Decision Making

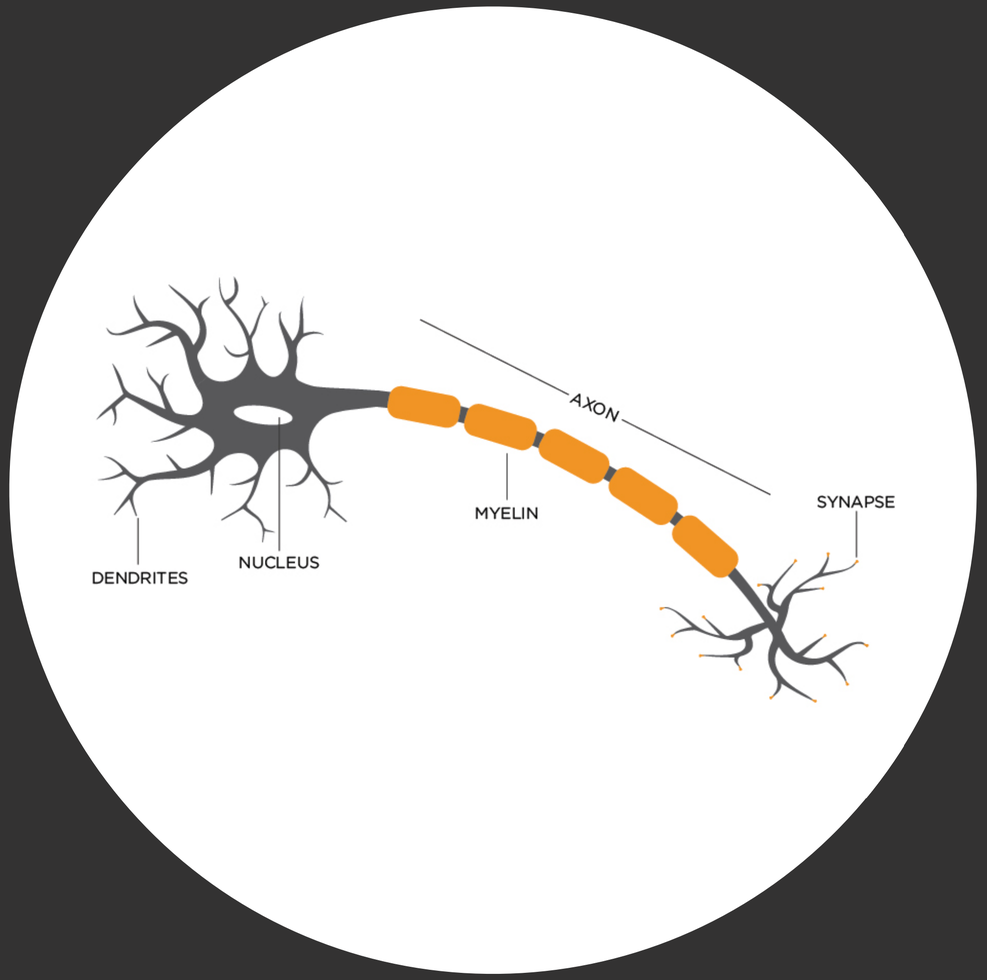


How does it work?

Biological Neuron

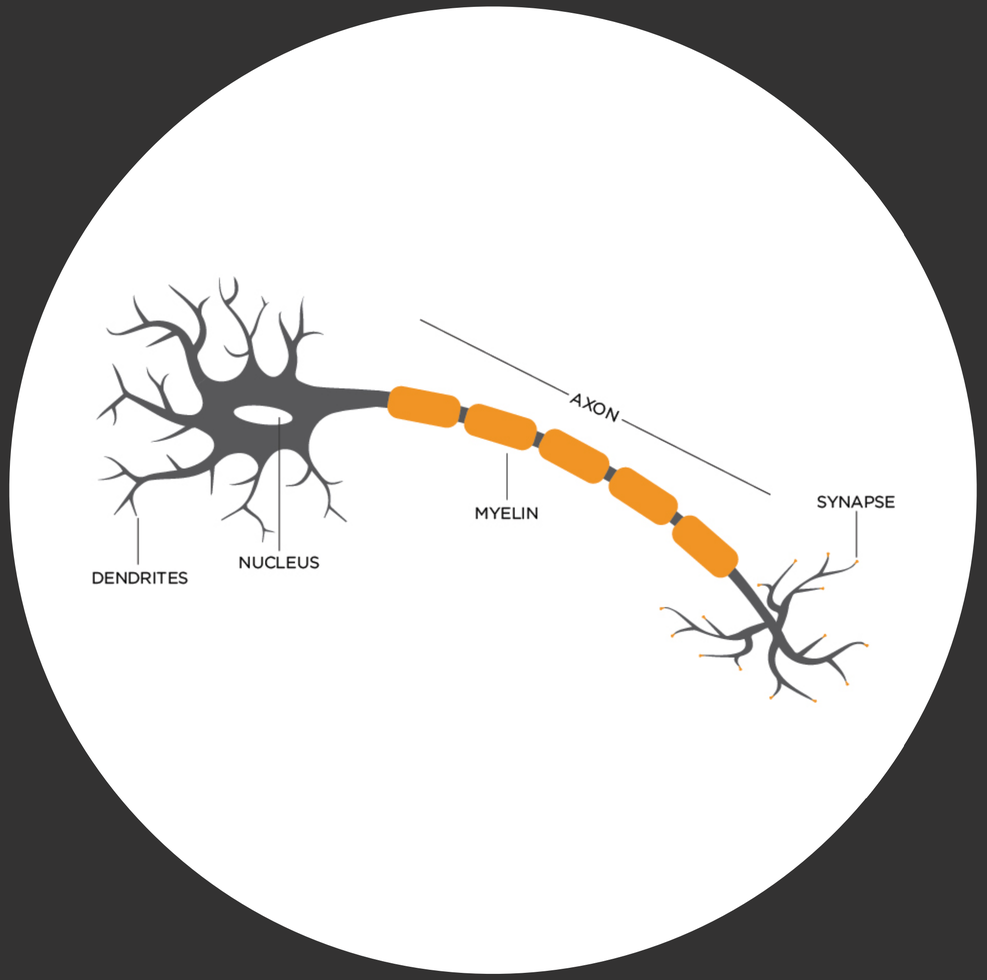
Artificial Neuron

Biological Neuron

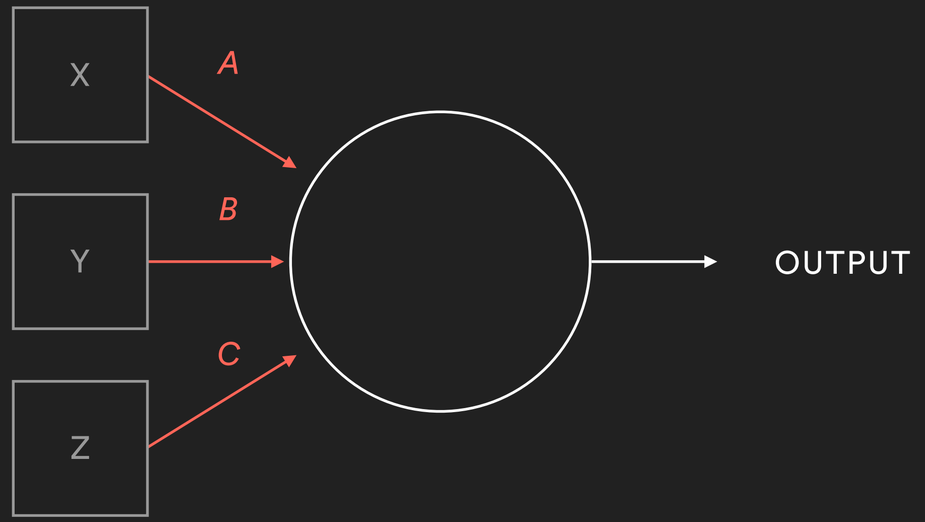


Artificial Neuron

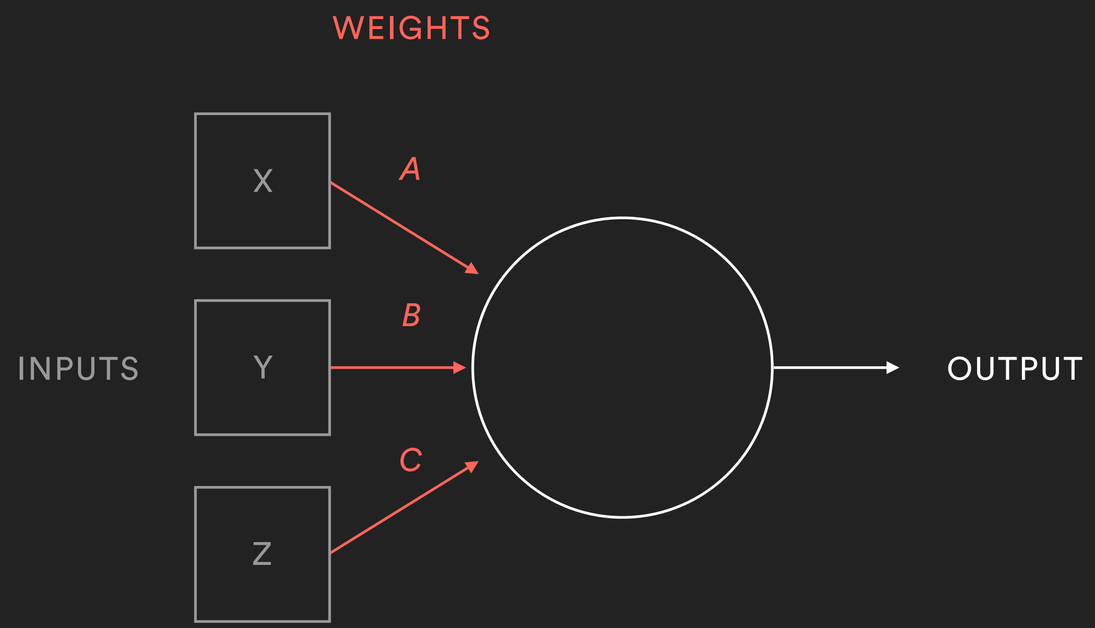
Biological Neuron



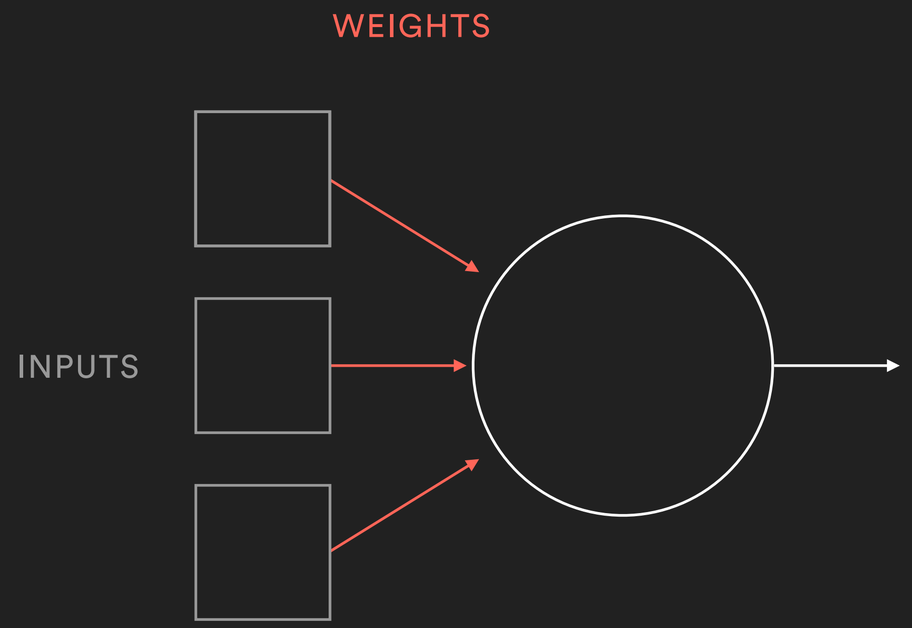
Artificial Neuron



Artificial Neuron

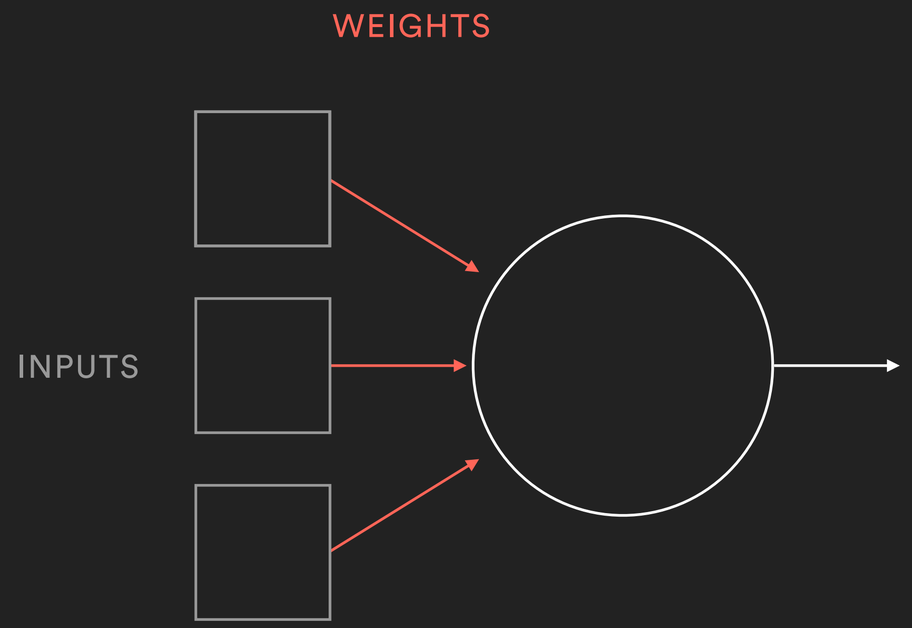


Artificial Neuron



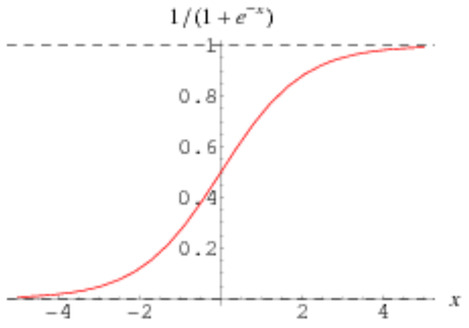
X *A* + Y *B* + Z *C* = OUTPUT

Artificial Neuron



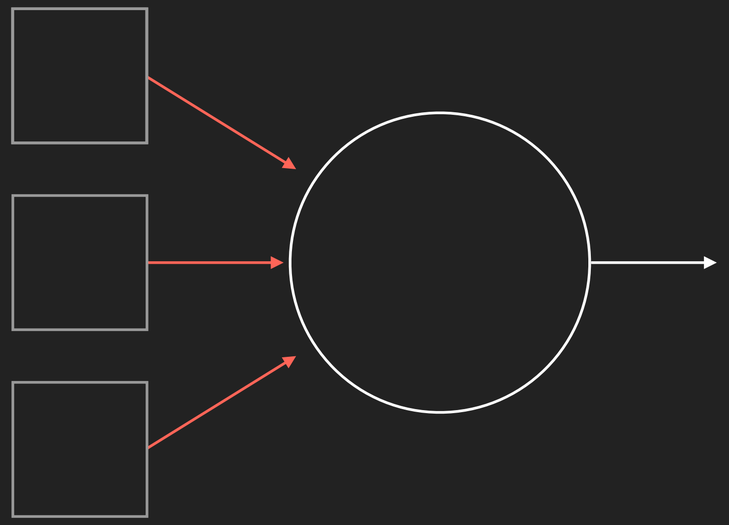
$$\sigma (X A + Y B + Z C) = \text{OUTPUT}$$

Artificial Neuron



WEIGHTS

INPUTS

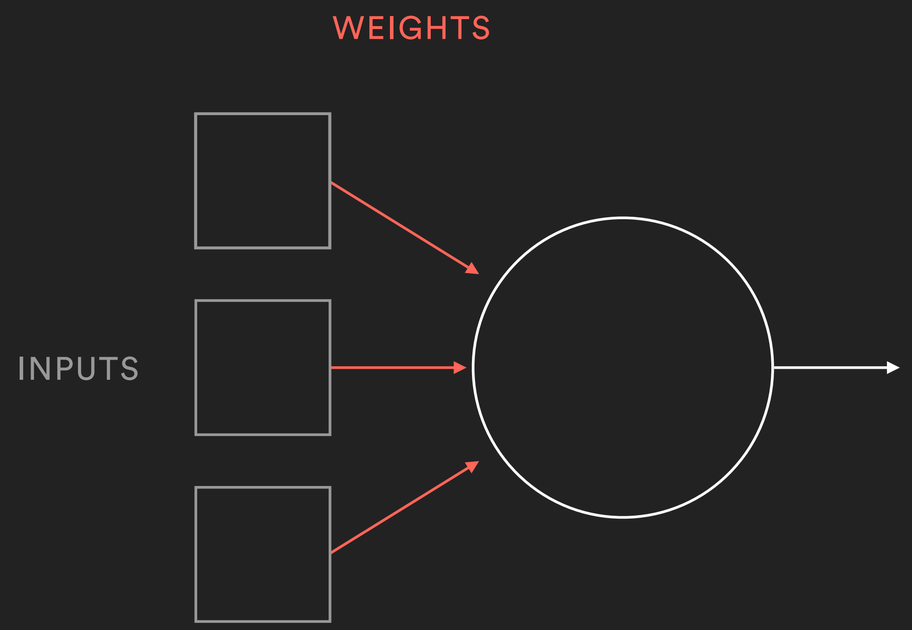


SIGMOID FUNCTION

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

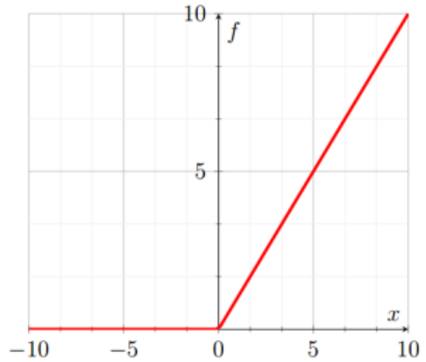
σ (X *A* + Y *B* + Z *C*) = OUTPUT

Artificial Neuron



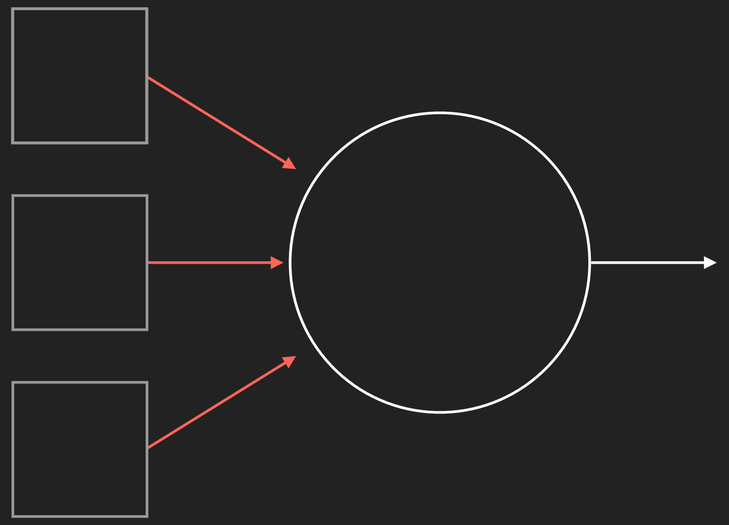
σ (X *A* + Y *B* + Z *C*) = OUTPUT

Artificial Neuron



WEIGHTS

INPUTS

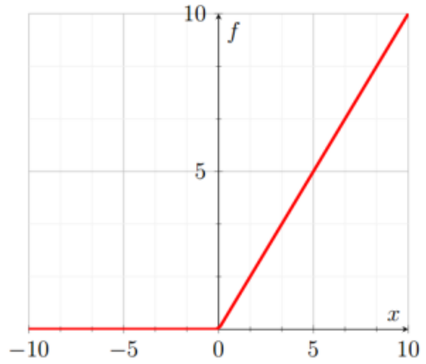


RELU FUNCTION

$$f(x) = x^+ = \max(0, x)$$

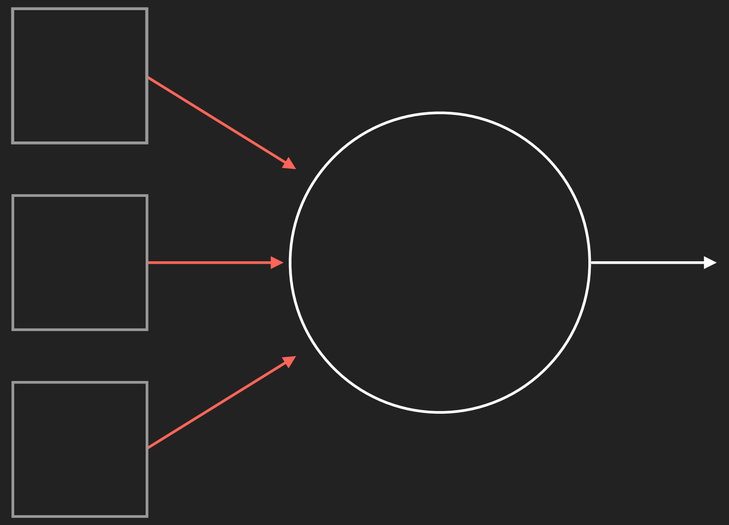
σ (X *A* + Y *B* + Z *C*) = OUTPUT

Artificial Neuron



WEIGHTS

INPUTS

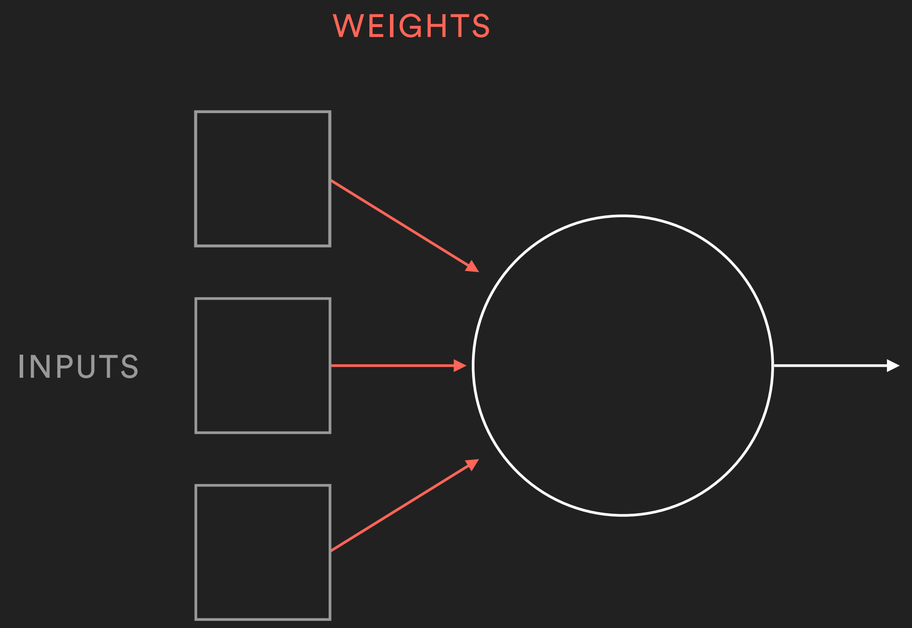


RELU FUNCTION

$$f(x) = x^+ = \max(0, x)$$

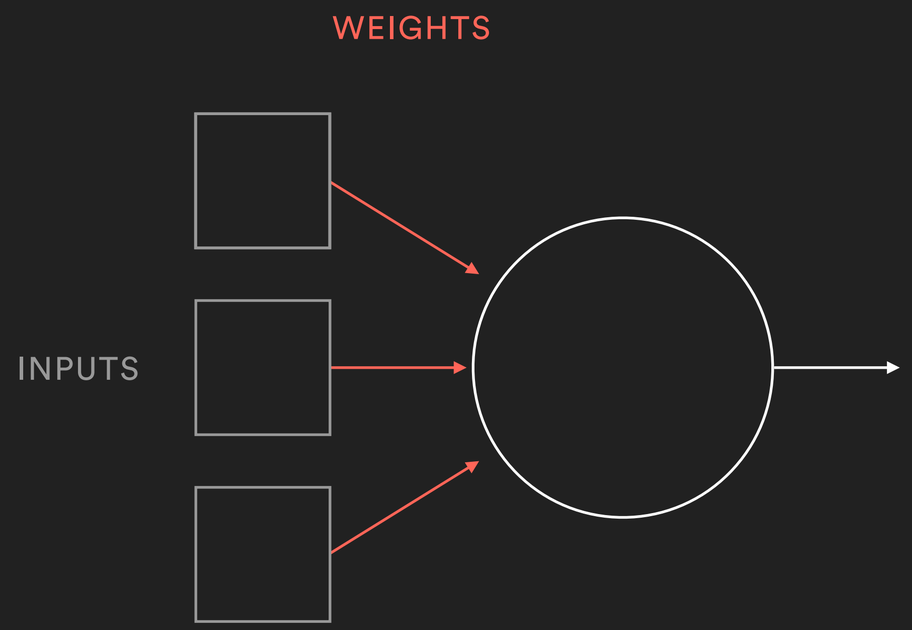
R (X *A* + Y *B* + Z *C*) = OUTPUT

Artificial Neuron



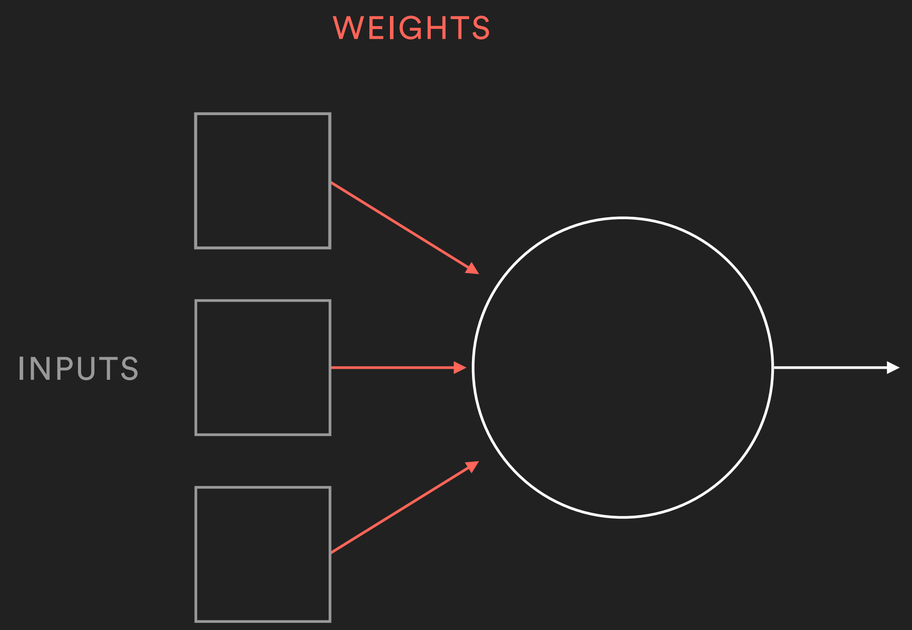
R (X *A* + Y *B* + Z *C*) = OUTPUT

Artificial Neuron



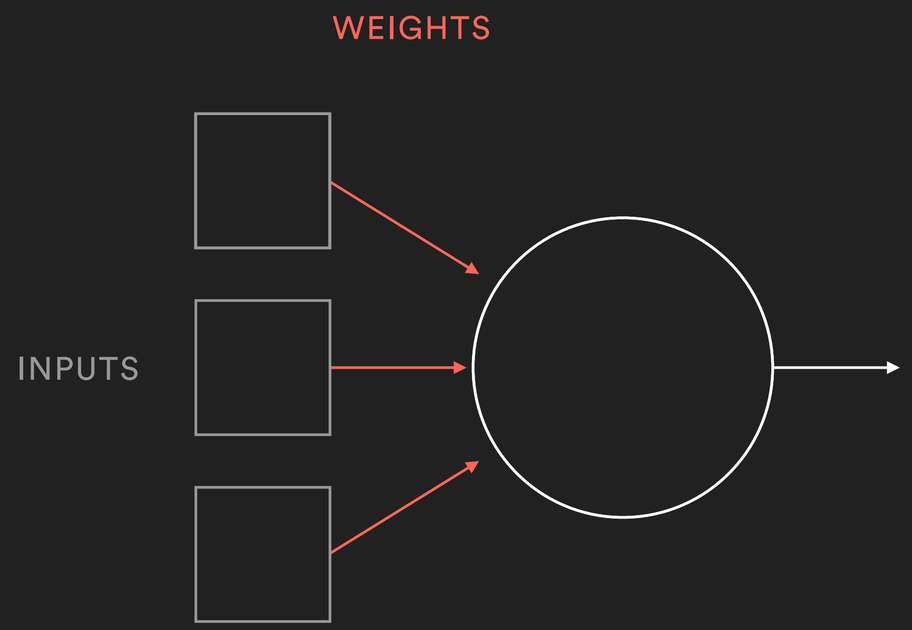
R (X *A* + Y *B* + Z *C*) = OUTPUT

Artificial Neuron



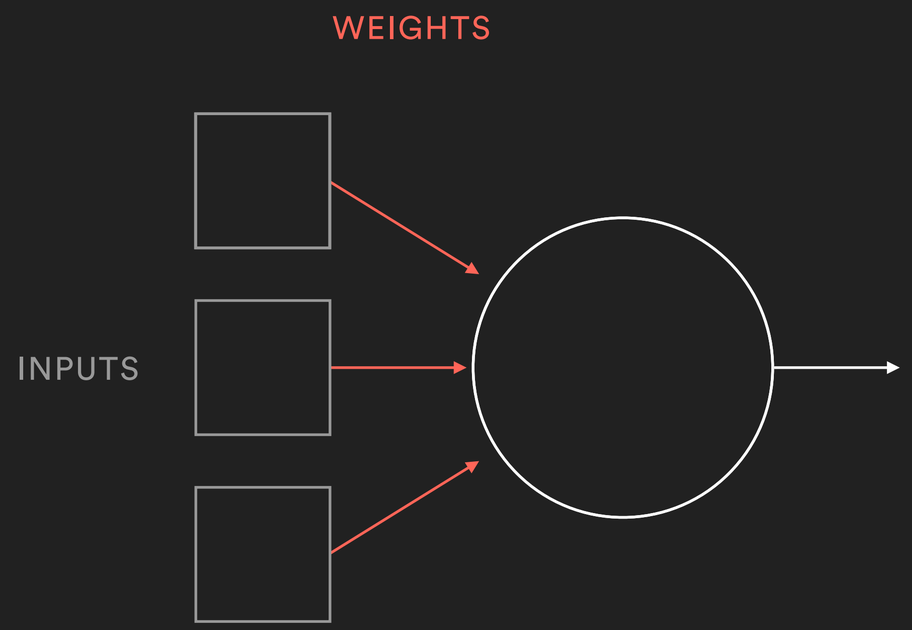
R (X *A* + Y *B* + Z *C* + BIAS) = OUTPUT

Artificial Neuron



R (X *A* + Y *B* + Z *C* + BIAS) = OUTPUT

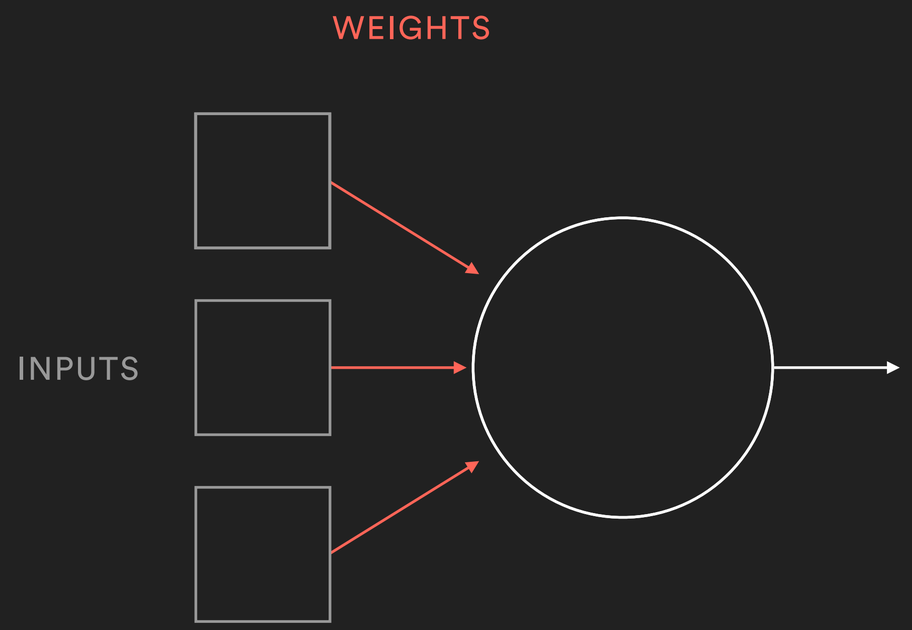
Artificial Neuron



R (X *A* + Y *B* + Z *C* + BIAS) = OUTPUT

= ACTIVATION FUNCTION

Artificial Neuron



R (X *A* + Y *B* + Z *C* + BIAS) = OUTPUT

= ACTIVATION FUNCTION

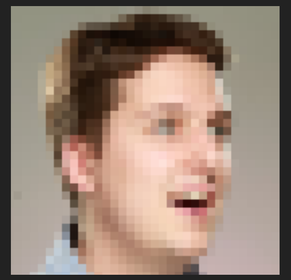
Artificial Neuron

ACTIVATED NOT ACTIVATED

BASED ON THE INPUT VALUE, THE WEIGHTS, AND THE BIAS

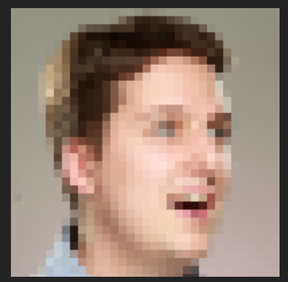
Building the network

INPUT LAYER OF ANN



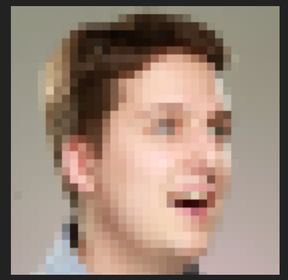
2D ARRAY OF
PIXEL VALUES

INPUT LAYER OF ANN



2D ARRAY OF
PIXEL VALUES

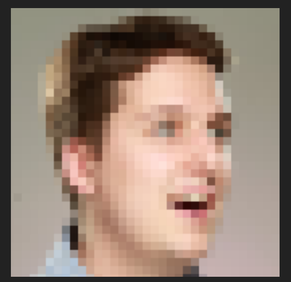
INPUT LAYER OF ANN



2D ARRAY OF
PIXEL VALUES

INPUT LAYER OF ANN

HIDDEN LAYERS OF NEURONS

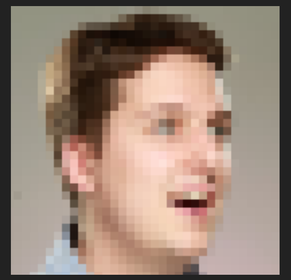


2D ARRAY OF
PIXEL VALUES

INPUT LAYER OF ANN

HIDDEN LAYERS OF NEURONS

OUTPUT



2D ARRAY OF
PIXEL VALUES

ANDRE

GAVIN

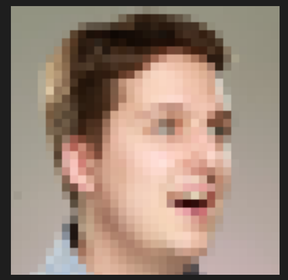
RYAN

BEN

INPUT LAYER OF ANN

HIDDEN LAYERS OF NEURONS

OUTPUT



2D ARRAY OF
PIXEL VALUES

ANDRE

GAVIN

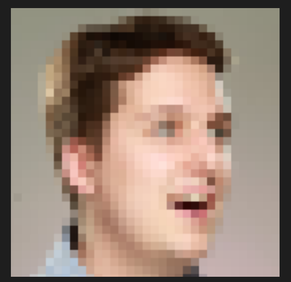
RYAN

BEN

INPUT LAYER OF ANN

HIDDEN LAYERS OF NEURONS

OUTPUT



2D ARRAY OF
PIXEL VALUES

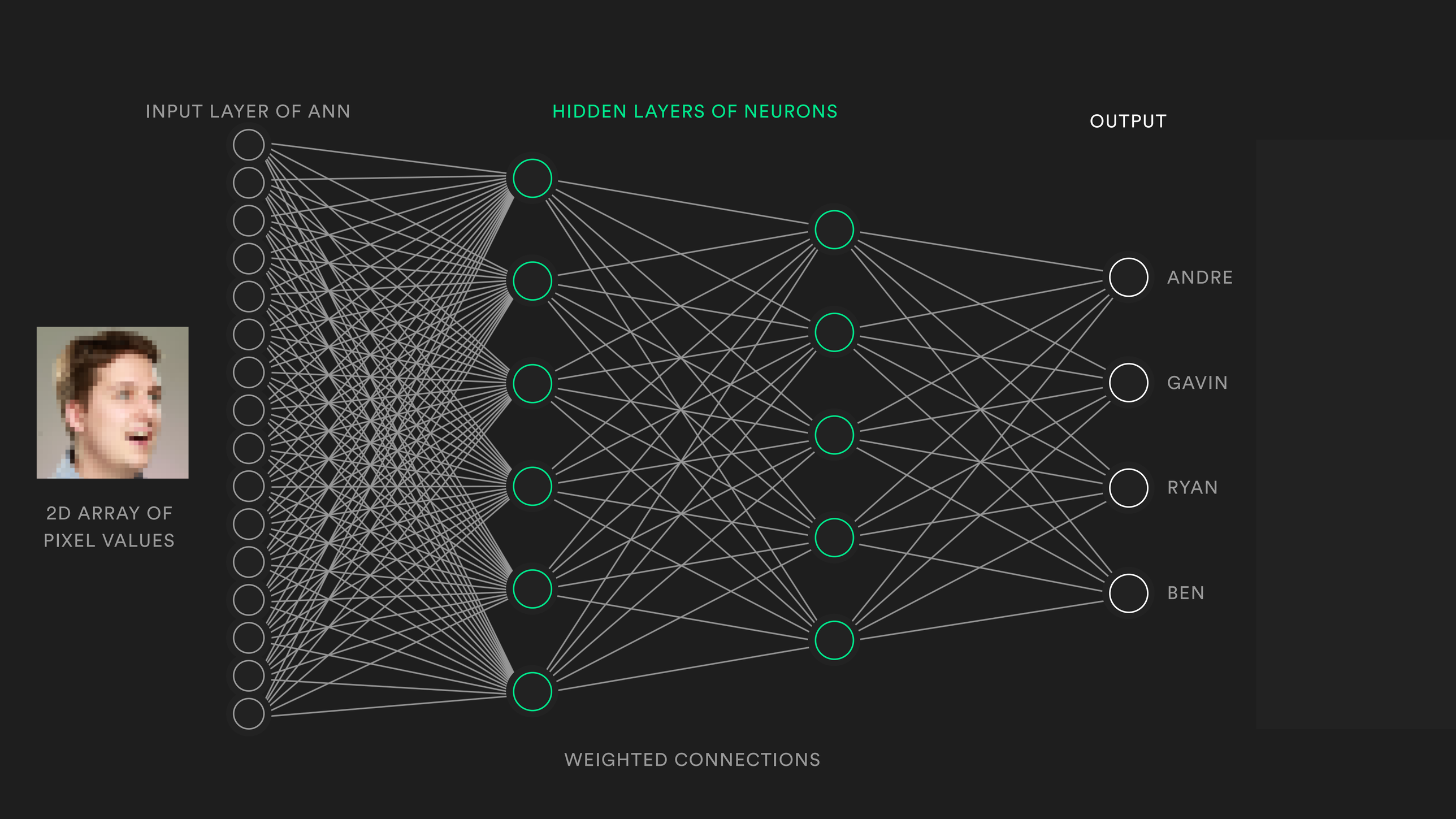
ANDRE

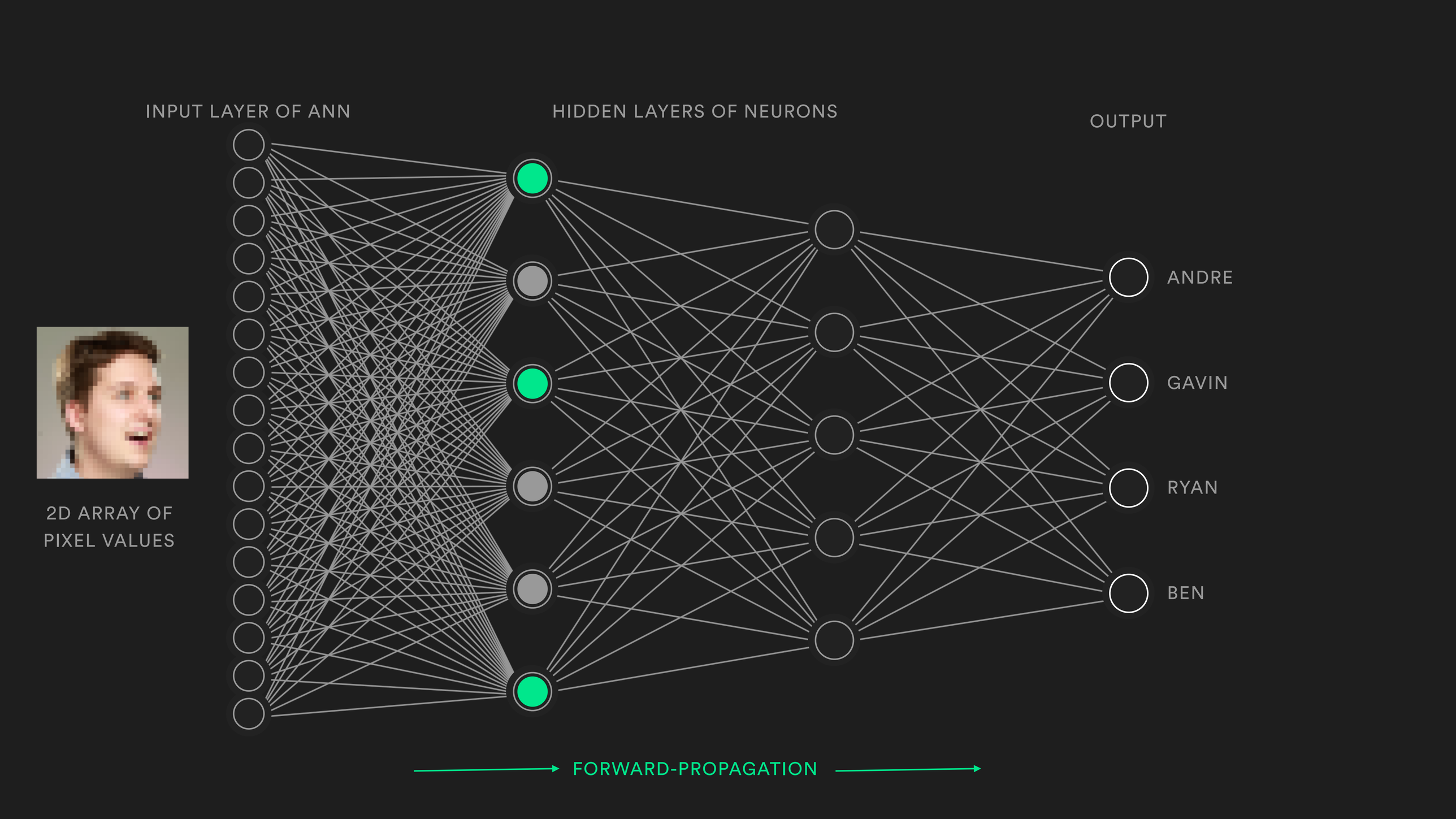
GAVIN

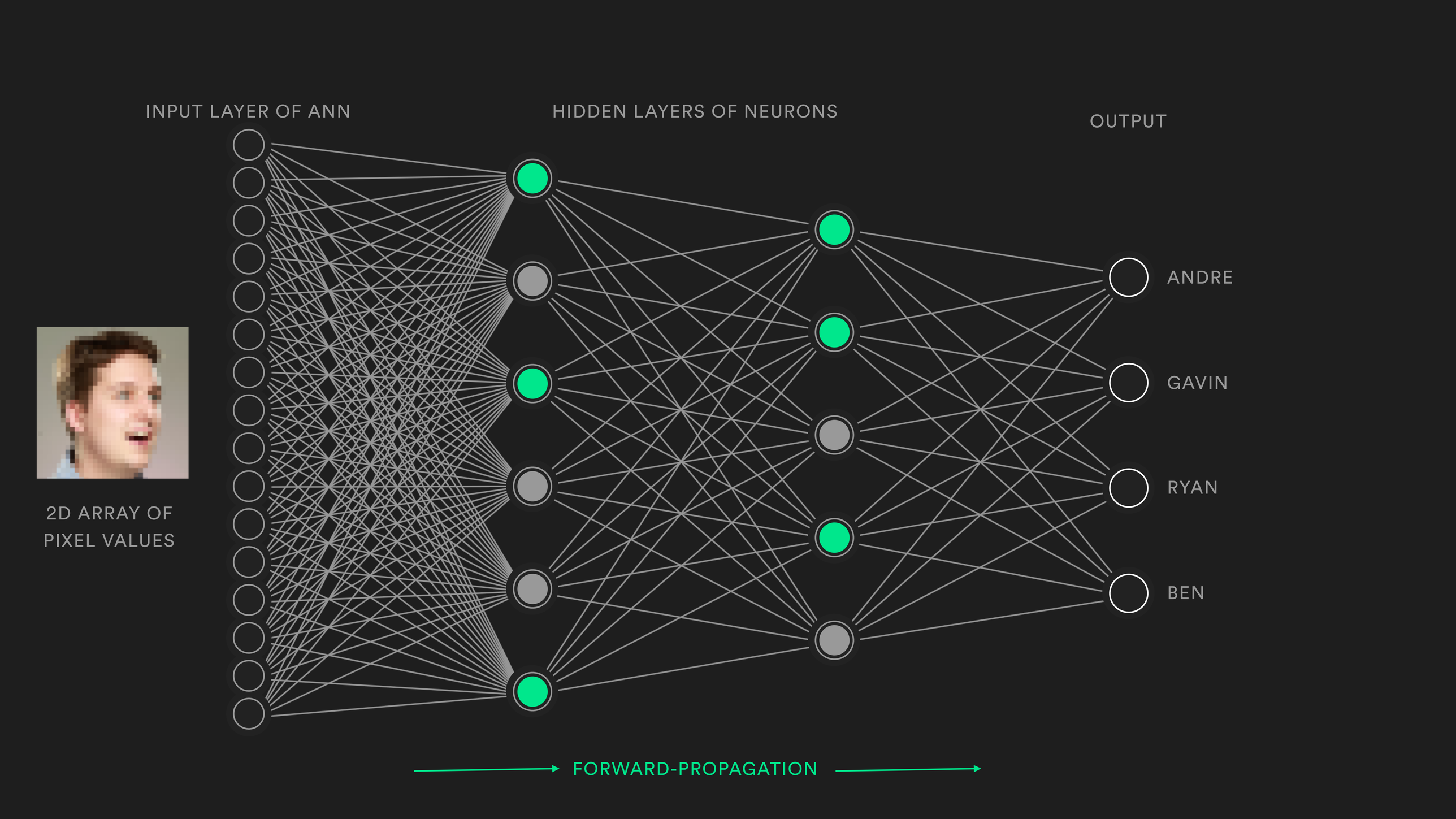
RYAN

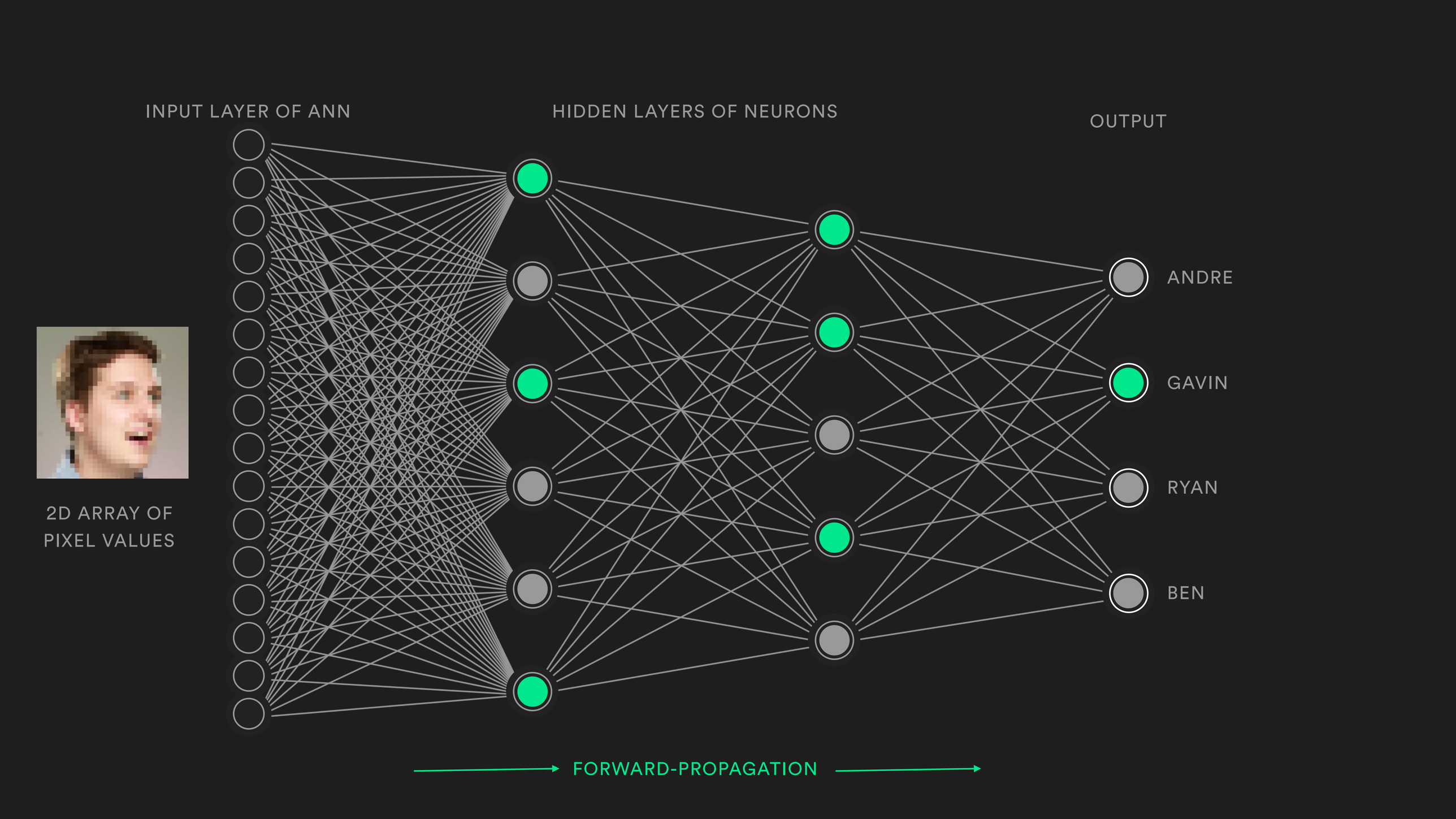
BEN

WEIGHTED CONNECTIONS

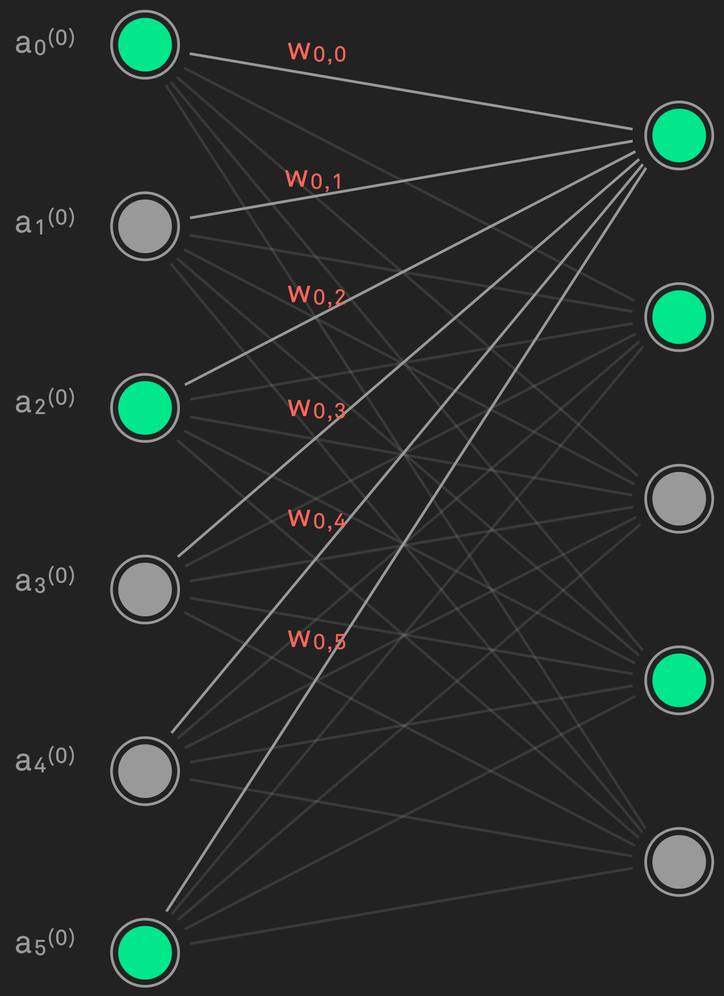

INPUT LAYER OF ANN
HIDDEN LAYERS OF NEURONS
OUTPUT
ANDRE
GAVIN
RYAN
BEN
2D ARRAY OF
PIXEL VALUES
WEIGHTED CONNECTIONS


INPUT LAYER OF ANN
HIDDEN LAYERS OF NEURONS
OUTPUT
ANDRE
GAVIN
RYAN
BEN
2D ARRAY OF
PIXEL VALUES
FORWARD-PROPAGATION

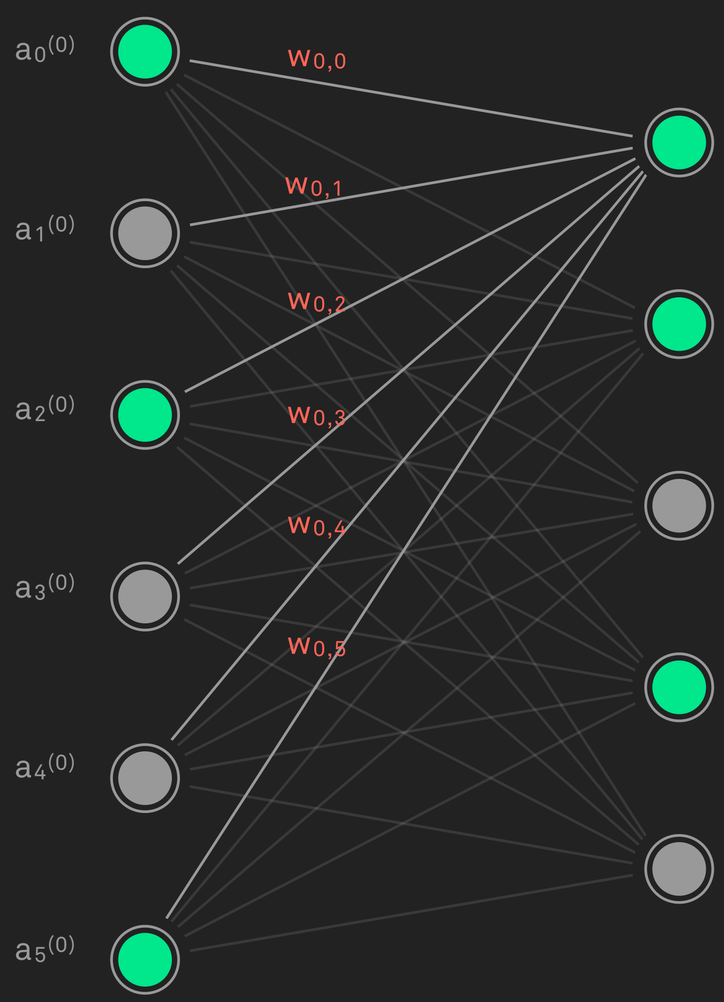

INPUT LAYER OF ANN
HIDDEN LAYERS OF NEURONS
OUTPUT
ANDRE
GAVIN
RYAN
BEN
2D ARRAY OF
PIXEL VALUES
FORWARD-PROPAGATION


INPUT LAYER OF ANN
HIDDEN LAYERS OF NEURONS
OUTPUT
ANDRE
GAVIN
RYAN
BEN
2D ARRAY OF
PIXEL VALUES
FORWARD-PROPAGATION

Simplified Notation



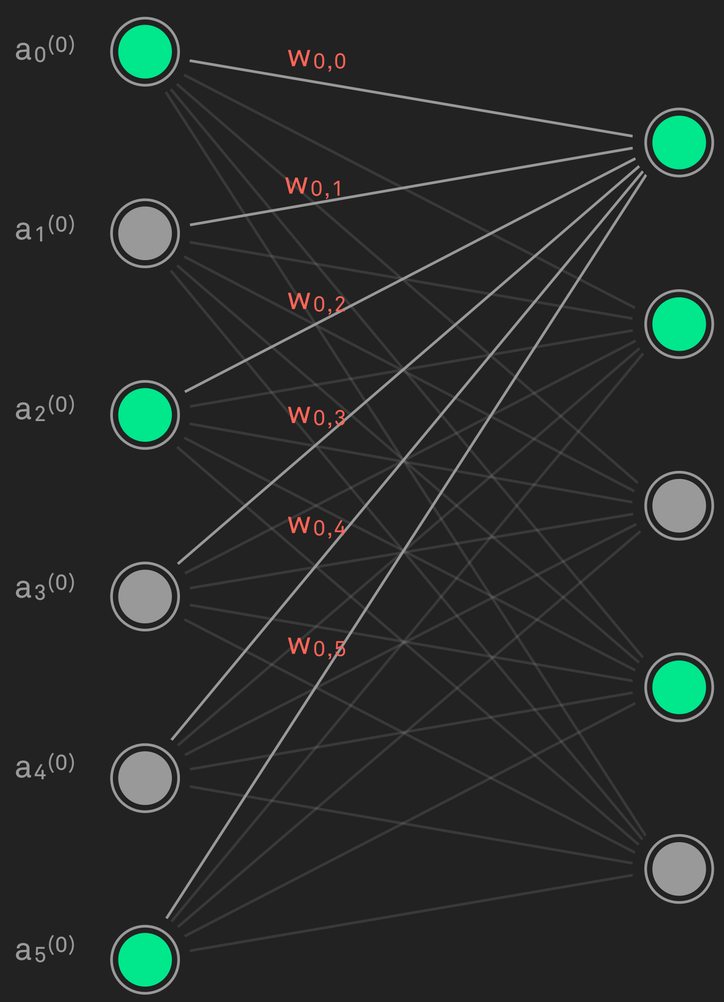
Simplified Notation



SIGMOID

$$\sigma\left(a_0^{(0)}w_{0,0} + a_1^{(0)}w_{0,1} + a_2^{(0)}w_{0,2} + a_3^{(0)}w_{0,3} + a_4^{(0)}w_{0,4} + a_5^{(0)}w_{0,5} + \text{bias}\right) = \text{Output}$$

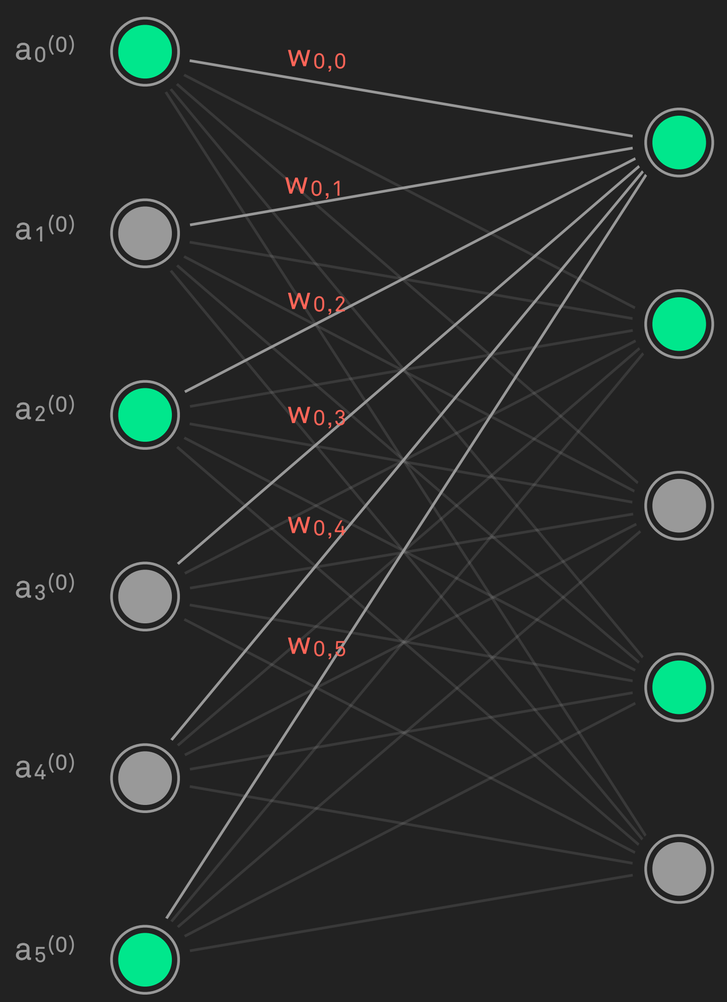
Simplified Notation



SIGMOID

$$\sigma\left(a_0^{(0)}w_{0,0} + a_1^{(0)}w_{0,1} + a_2^{(0)}w_{0,2} + a_3^{(0)}w_{0,3} + a_4^{(0)}w_{0,4} + a_5^{(0)}w_{0,5} + \text{bias}\right) = \text{Output}$$

Simplified Notation

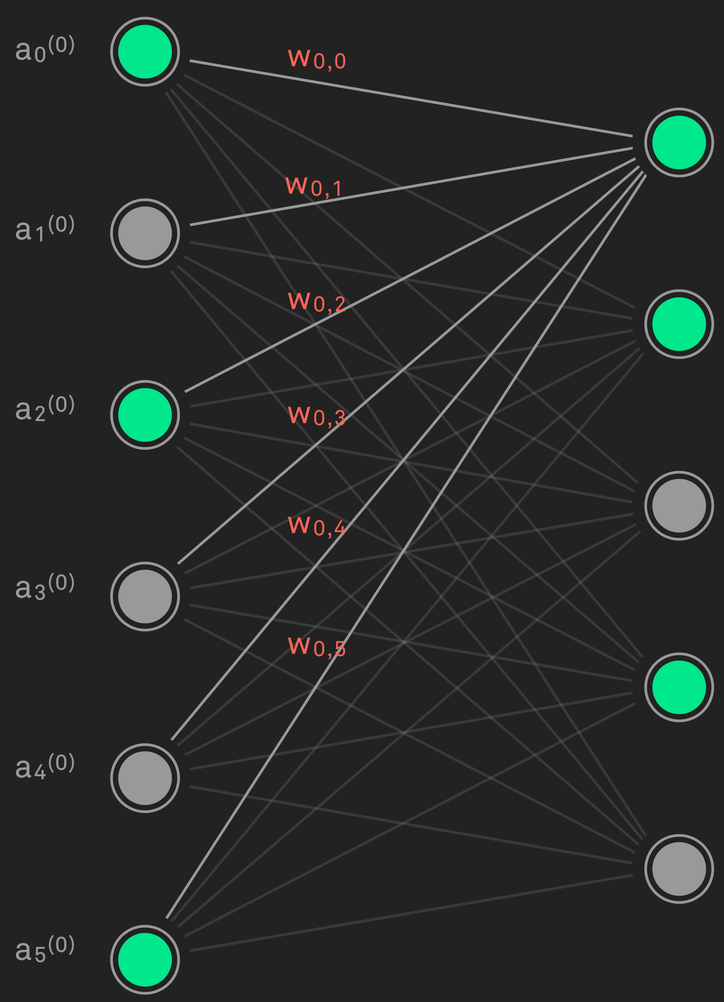


SIGMOID

$$\sigma\left(a_0^{(0)}w_{0,0} + a_1^{(0)}w_{0,1} + a_2^{(0)}w_{0,2} + a_3^{(0)}w_{0,3} + a_4^{(0)}w_{0,4} + a_5^{(0)}w_{0,5} + \text{bias}\right) = \text{Output}$$

$$\begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix}$$

Simplified Notation

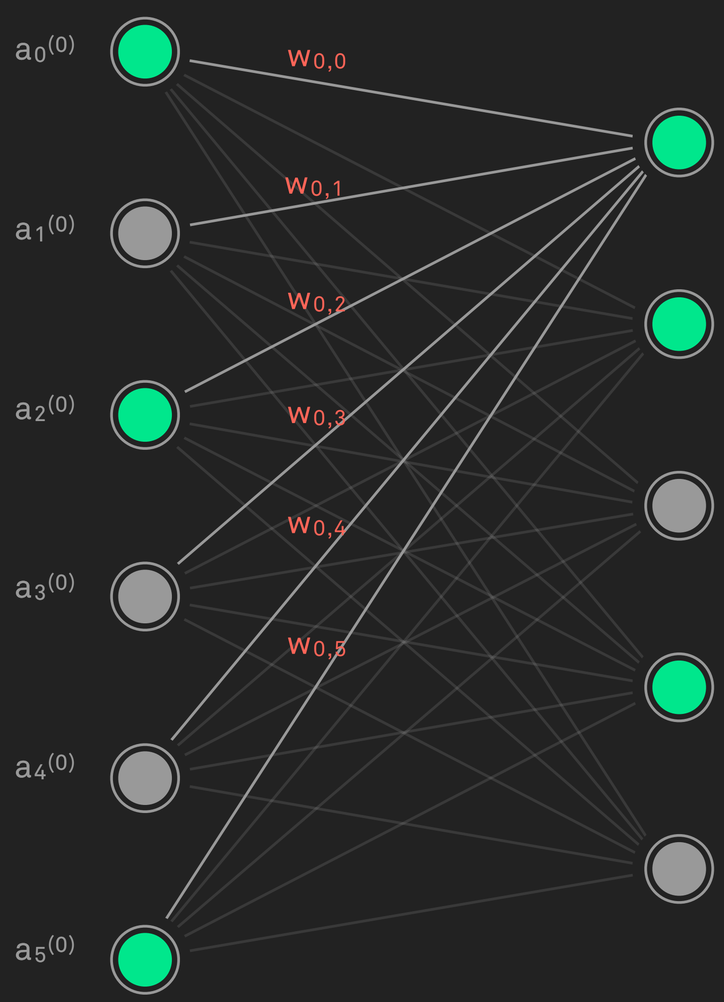


SIGMOID

$$\sigma\left(a_0^{(0)}w_{0,0} + a_1^{(0)}w_{0,1} + a_2^{(0)}w_{0,2} + a_3^{(0)}w_{0,3} + a_4^{(0)}w_{0,4} + a_5^{(0)}w_{0,5} + \text{bias}\right) = \text{Output}$$

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} & w_{0,5} \\ & & \dots & & & \\ & & \dots & & & \\ & & \dots & & & \\ & & \dots & & & \\ w_{k,0} & w_{k,1} & w_{k,2} & w_{k,3} & w_{k,4} & w_{k,5} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix}$$

Simplified Notation

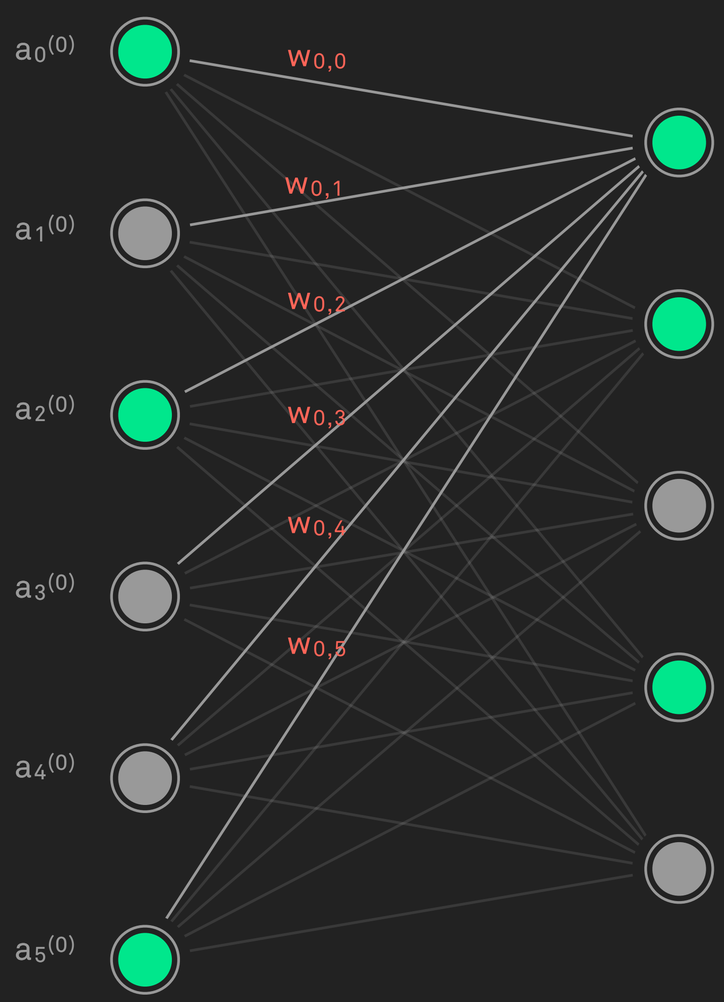


SIGMOID

$$\sigma\left(a_0^{(0)}w_{0,0} + a_1^{(0)}w_{0,1} + a_2^{(0)}w_{0,2} + a_3^{(0)}w_{0,3} + a_4^{(0)}w_{0,4} + a_5^{(0)}w_{0,5} + \text{bias}\right) = \text{Output}$$

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} & w_{0,5} \\ & & \dots & & & \\ & & \dots & & & \\ & & \dots & & & \\ & & \dots & & & \\ w_{k,0} & w_{k,1} & w_{k,2} & w_{k,3} & w_{k,4} & w_{k,5} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix}$$

Simplified Notation

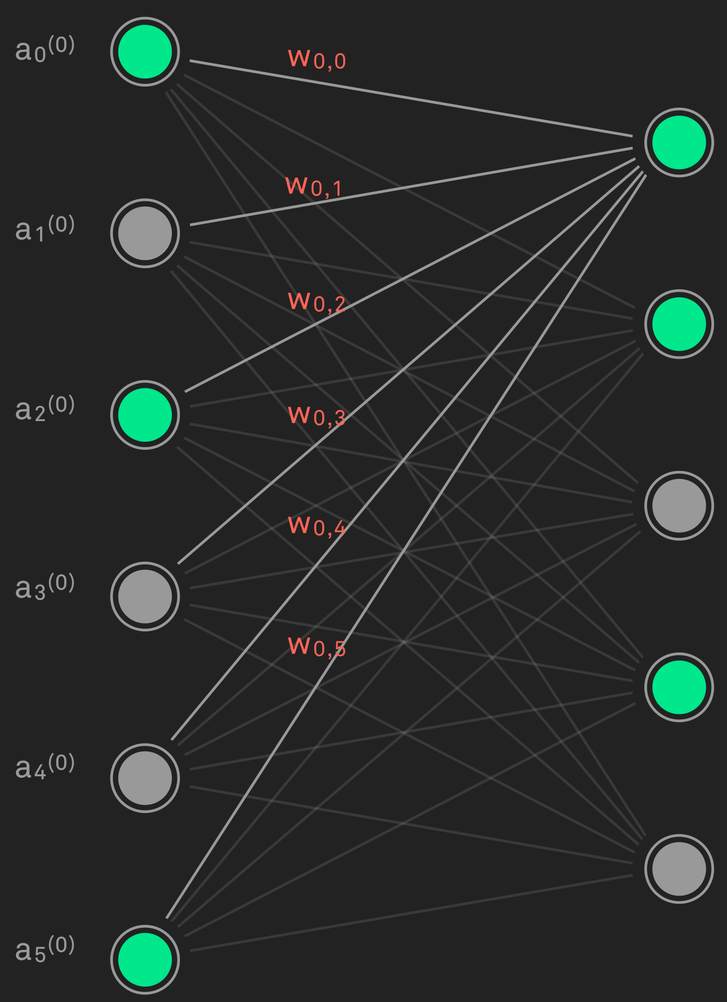


SIGMOID

$$\sigma\left(a_0^{(0)}w_{0,0} + a_1^{(0)}w_{0,1} + a_2^{(0)}w_{0,2} + a_3^{(0)}w_{0,3} + a_4^{(0)}w_{0,4} + a_5^{(0)}w_{0,5} + \text{bias}\right) = \text{Output}$$

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} & w_{0,5} \\ & & \dots & & & \\ & & \dots & & & \\ & & \dots & & & \\ & & \dots & & & \\ w_{k,0} & w_{k,1} & w_{k,2} & w_{k,3} & w_{k,4} & w_{k,5} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix} + \begin{bmatrix} b_1 \\ \dots \\ \dots \\ \dots \\ \dots \\ b_n \end{bmatrix}$$

Simplified Notation

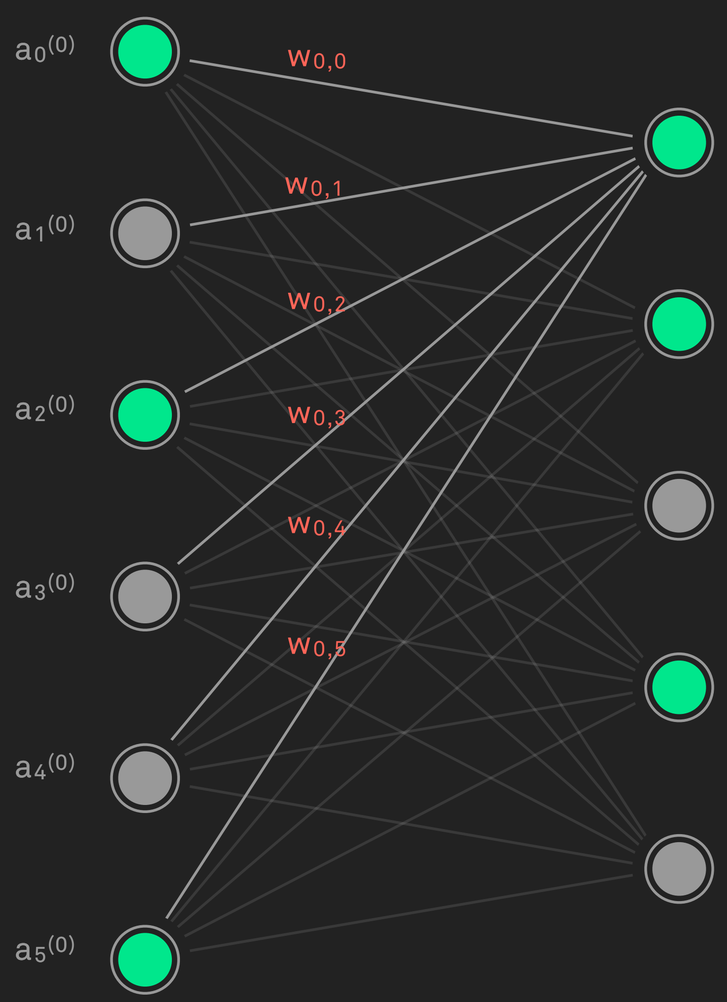


SIGMOID

$$\sigma\left(a_0^{(0)}w_{0,0} + a_1^{(0)}w_{0,1} + a_2^{(0)}w_{0,2} + a_3^{(0)}w_{0,3} + a_4^{(0)}w_{0,4} + a_5^{(0)}w_{0,5} + \text{bias}\right) = \text{Output}$$

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} & w_{0,5} \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ w_{k,0} & w_{k,1} & w_{k,2} & w_{k,3} & w_{k,4} & w_{k,5} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix} + \begin{bmatrix} b_1 \\ \dots \\ \dots \\ \dots \\ \dots \\ b_n \end{bmatrix}\right)$$

Simplified Notation

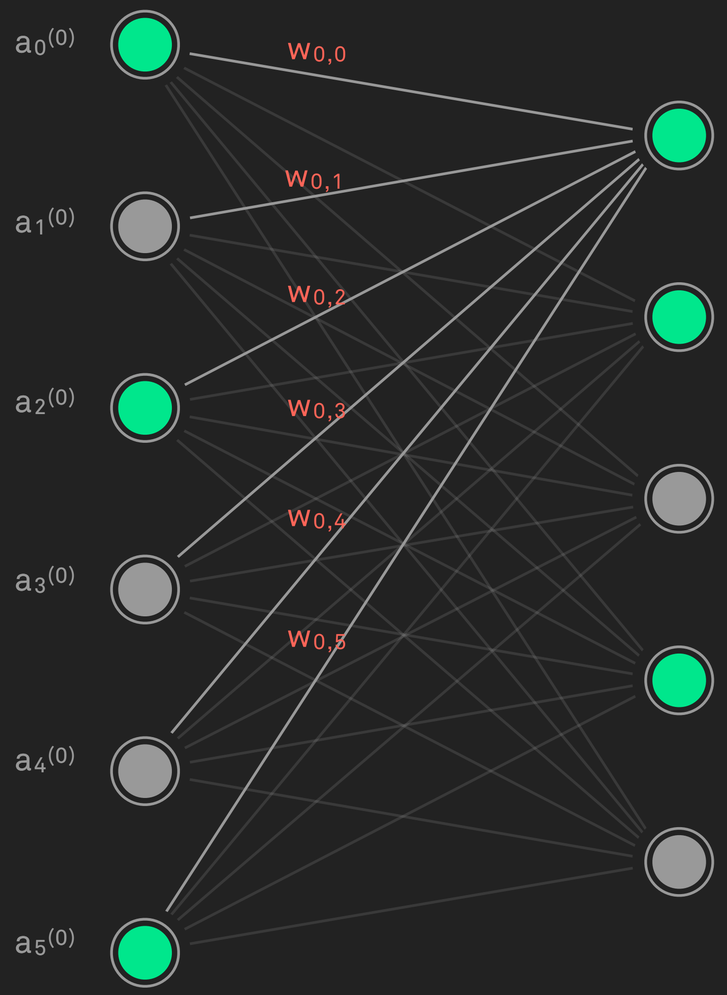


SIGMOID

$$\sigma\left(a_0^{(0)}w_{0,0} + a_1^{(0)}w_{0,1} + a_2^{(0)}w_{0,2} + a_3^{(0)}w_{0,3} + a_4^{(0)}w_{0,4} + a_5^{(0)}w_{0,5} + \text{bias}\right) = \text{Output}$$

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} & w_{0,5} \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ w_{k,0} & w_{k,1} & w_{k,2} & w_{k,3} & w_{k,4} & w_{k,5} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix} + \begin{bmatrix} b_1 \\ \dots \\ \dots \\ \dots \\ \dots \\ b_n \end{bmatrix}\right)$$

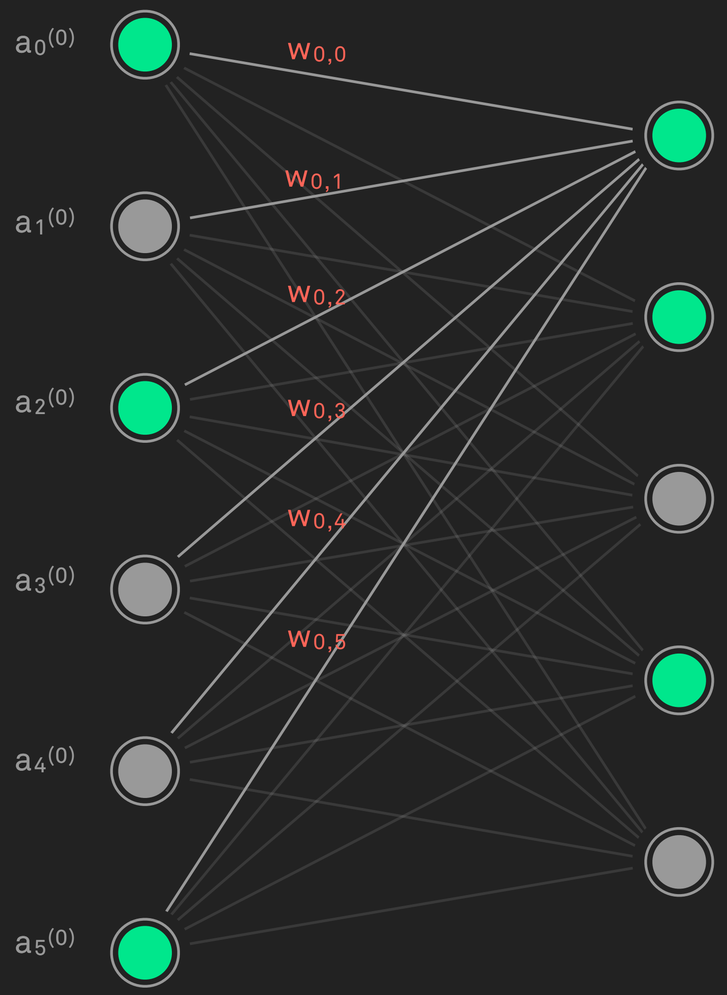
Simplified Notation



$$W$$

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} & w_{0,5} \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ w_{k,0} & w_{k,1} & w_{k,2} & w_{k,3} & w_{k,4} & w_{k,5} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix} + \begin{bmatrix} b_1 \\ \dots \\ \dots \\ \dots \\ \dots \\ b_n \end{bmatrix}\right)$$

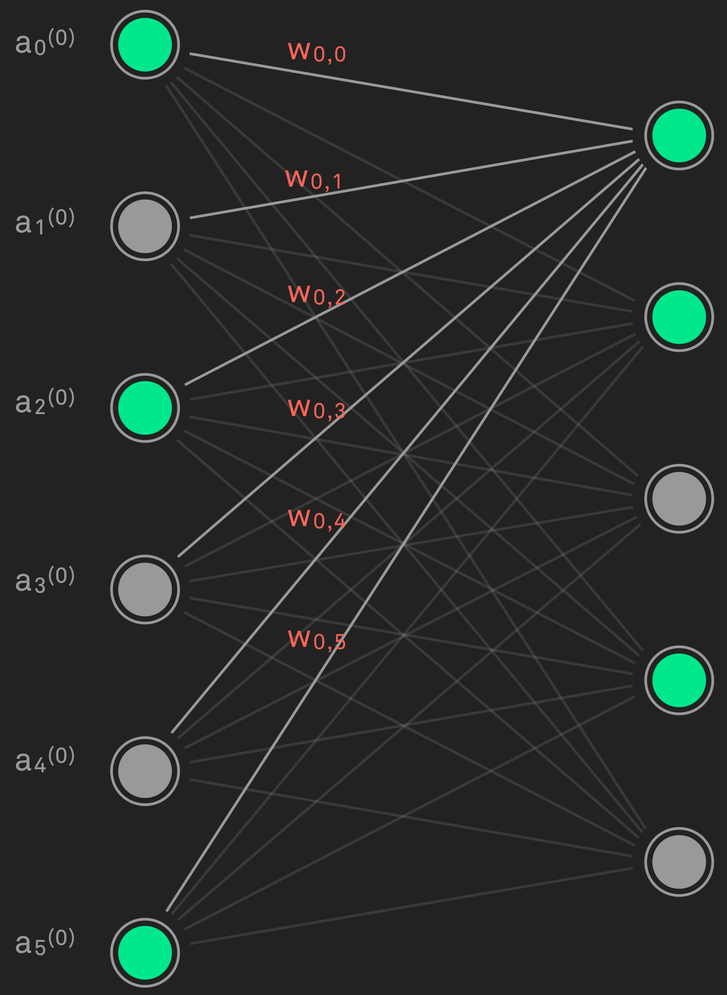
Simplified Notation



$$W\,a^{(0)}$$

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} & w_{0,5} \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ w_{k,0} & w_{k,1} & w_{k,2} & w_{k,3} & w_{k,4} & w_{k,5} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix} + \begin{bmatrix} b_1 \\ \dots \\ \dots \\ \dots \\ \dots \\ b_n \end{bmatrix}\right)$$

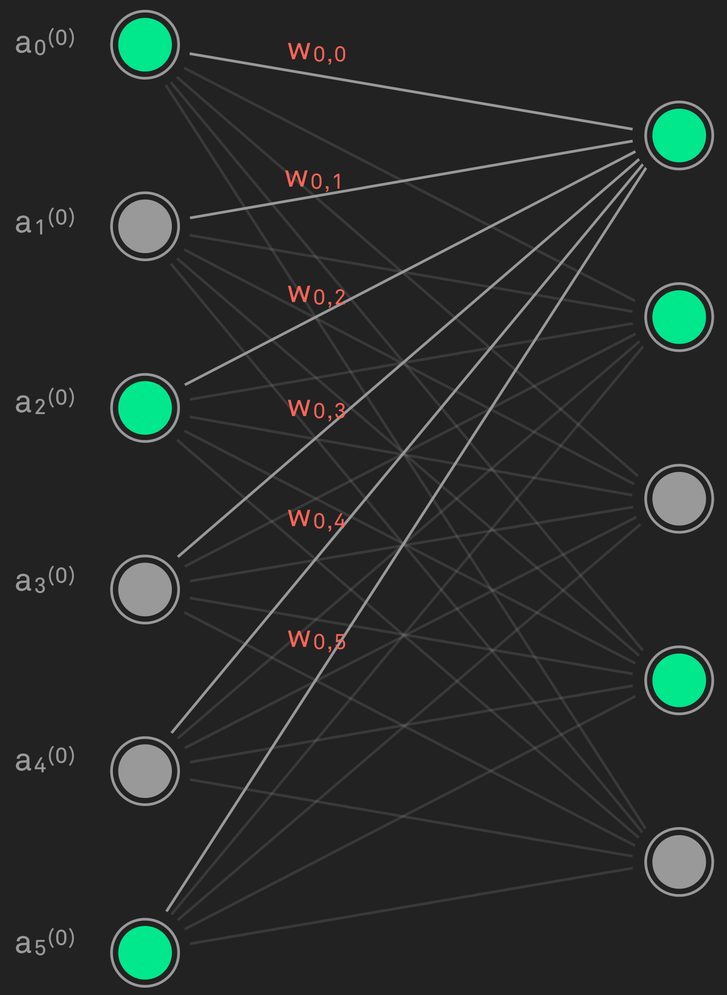
Simplified Notation



$$W\,a^{(0)} + B$$

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} & w_{0,5} \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ & & & \dots & & \\ w_{k,0} & w_{k,1} & w_{k,2} & w_{k,3} & w_{k,4} & w_{k,5} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix} + \begin{bmatrix} b_1 \\ \dots \\ \dots \\ \dots \\ \dots \\ b_n \end{bmatrix}\right)$$

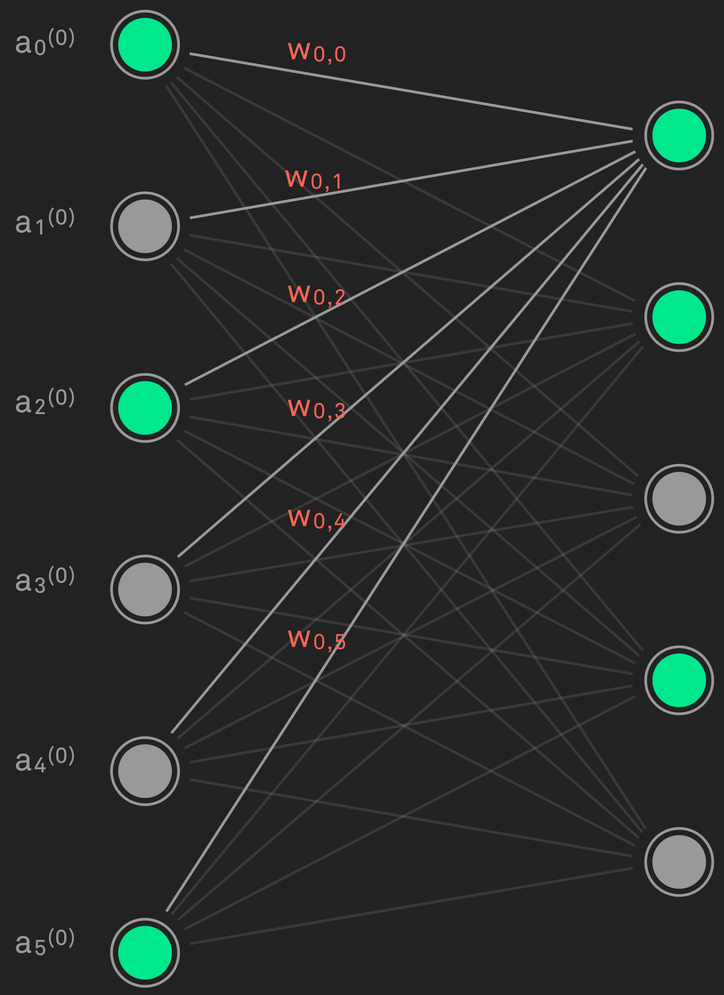
Simplified Notation



$$\sigma(Wa^{(0)} + B)$$

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} & w_{0,5} \\ \cdots & & & & & \\ \cdots & & & & & \\ \cdots & & & & & \\ \cdots & & & & & \\ w_{k,0} & w_{k,1} & w_{k,2} & w_{k,3} & w_{k,4} & w_{k,5} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \\ a_5^{(0)} \end{bmatrix} + \begin{bmatrix} b_1 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ b_n \end{bmatrix}\right)$$

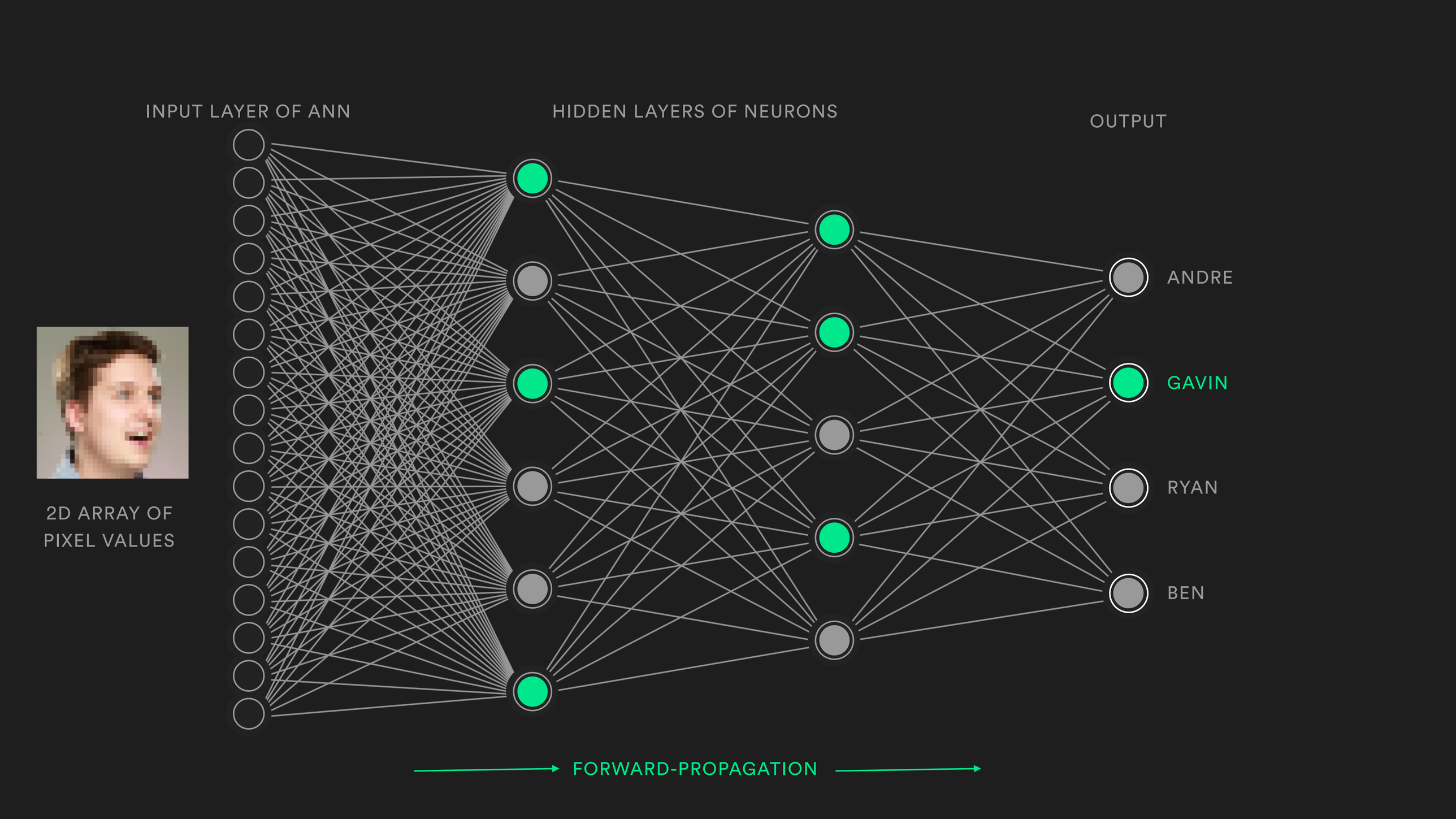
Simplified Notation



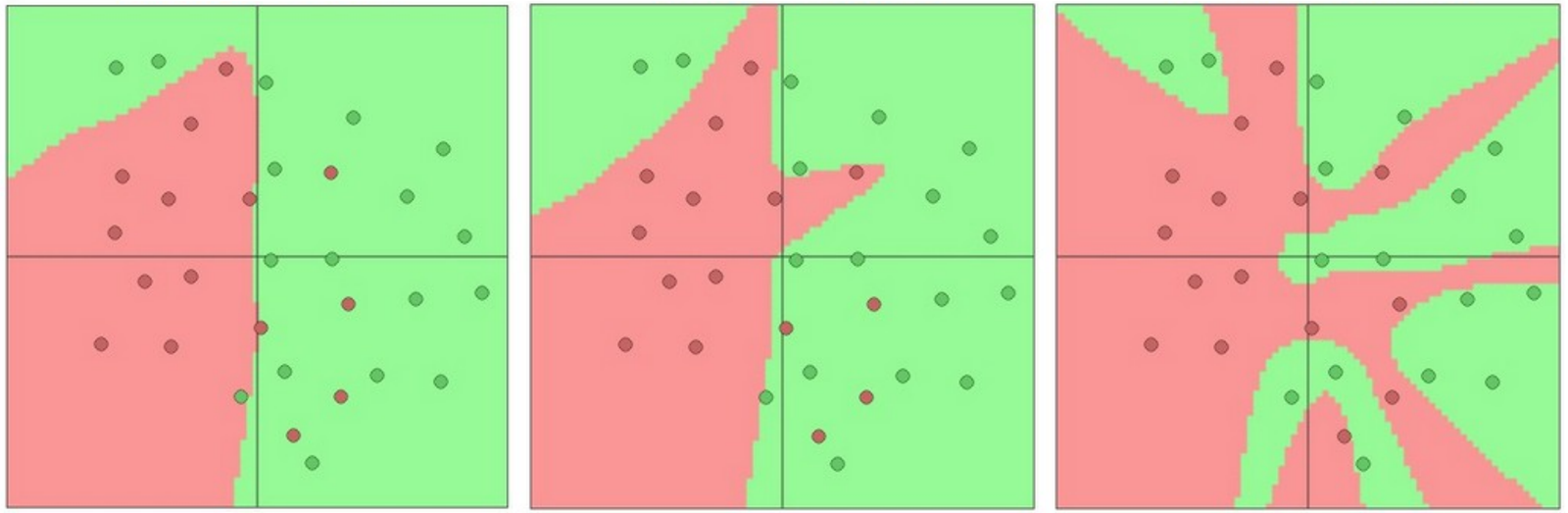
$$\sigma(W a^{(0)} + B)$$

```
class Network(object):
    def __init__(self, *args, **kwargs):
        #...yada yada, initialize weights and biases...

    def feedforward(self, a):
        """Return the output of the network for an input vector a"""
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a) + b)
        return a
```


INPUT LAYER OF ANN
HIDDEN LAYERS OF NEURONS
OUTPUT
ANDRE
GAVIN
RYAN
BEN
2D ARRAY OF
PIXEL VALUES
FORWARD-PROPAGATION

Larger Neural Networks can represent
more complicated functions.

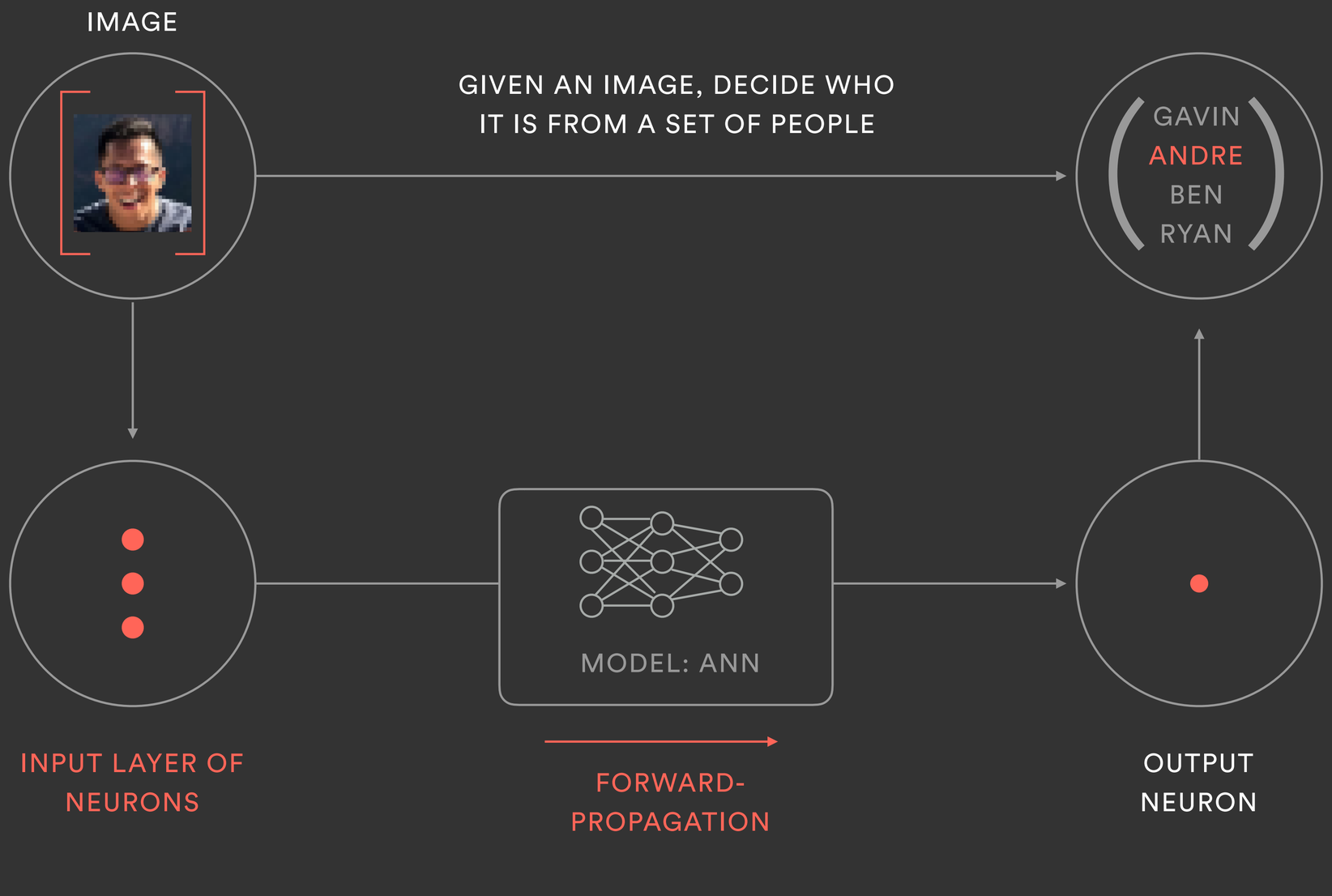


3
HIDDEN NEURONS

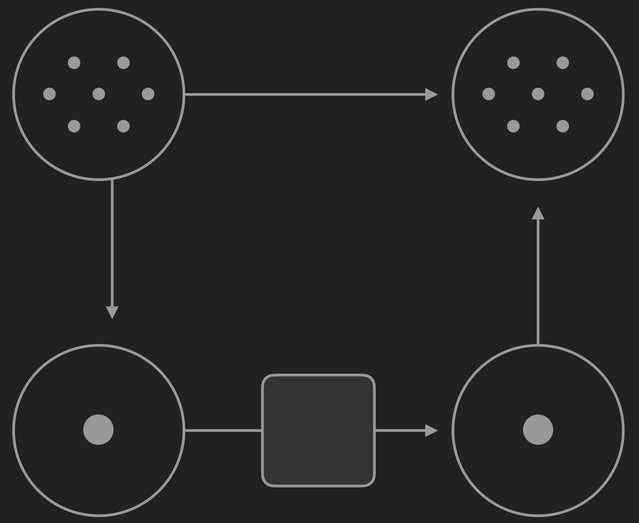
6
HIDDEN NEURONS

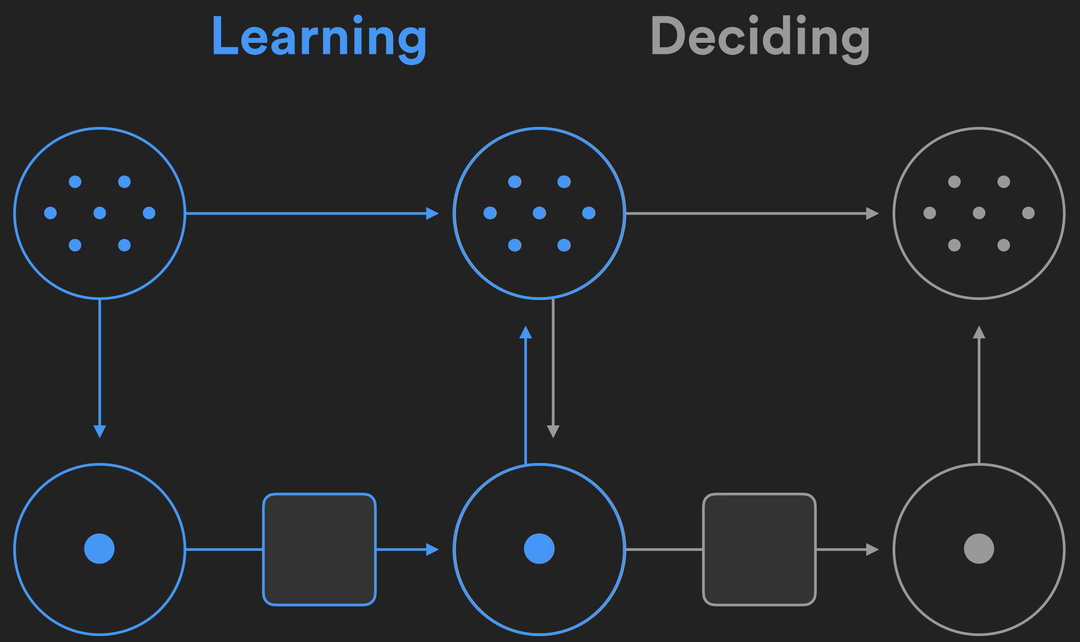
20
HIDDEN NEURONS

Decision Making

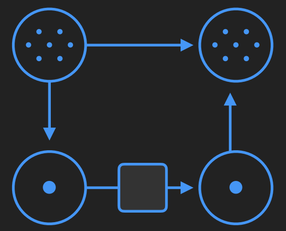


Deciding




Learning
Deciding

INTRODUCTION | DECIDING | LEARNING | CORRECTNESS



ANN Implementation Overview

MODEL

STEP 1

Define the input

STEP 2

Define the topology of the neural net
(i.e., the layers of neurons and their connections)

TRAINING
THE MODEL

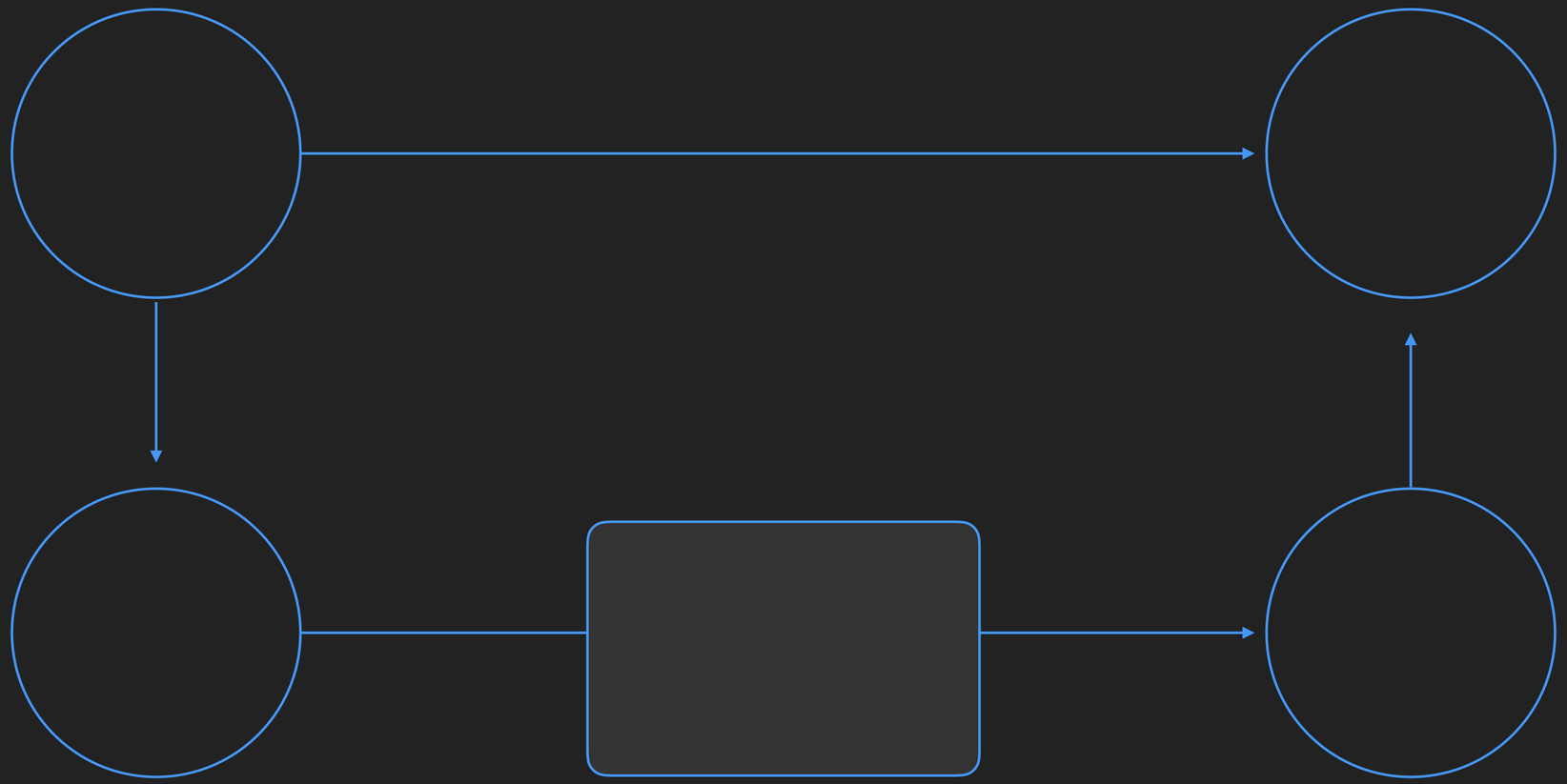
STEP 3

Train the neural net on examples of the problem

STEP 4

Run the trained neural net to solve new examples
of the problem

Learning



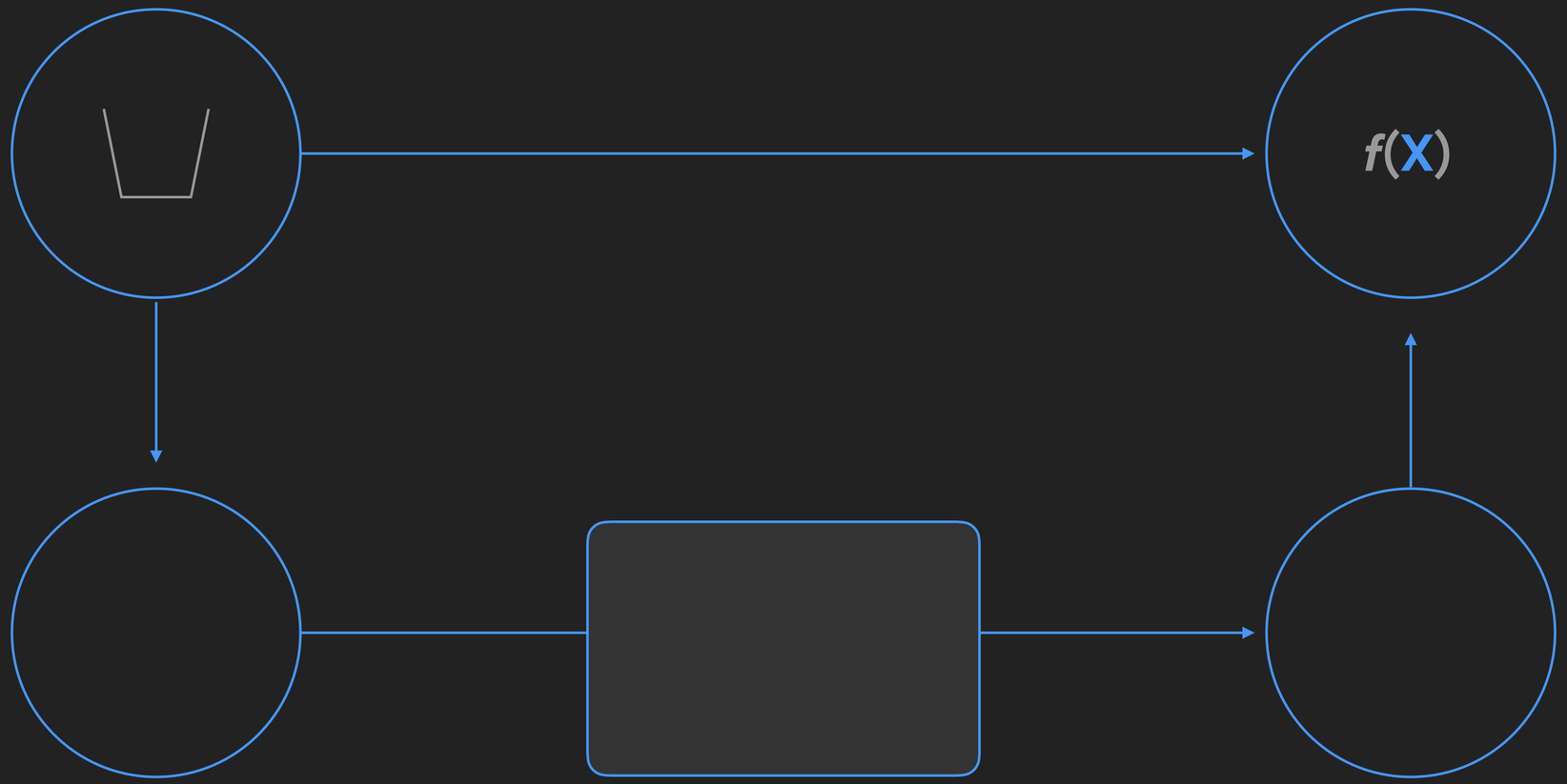
Learning

TRAINING DATA

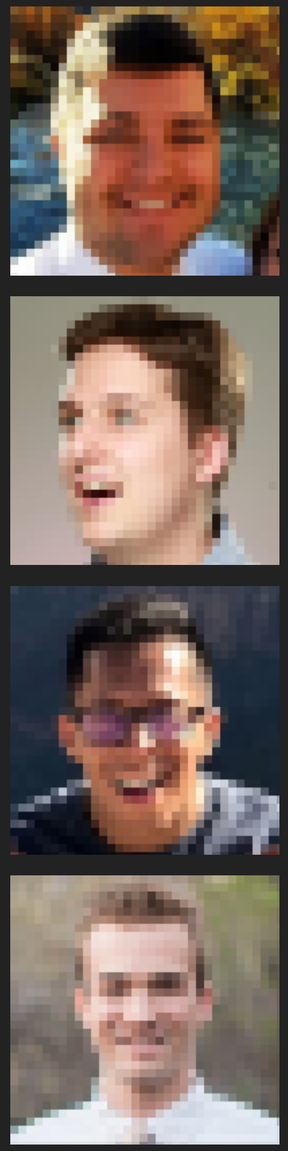
BUCKET OF MODELS

NORMS (COST FUNCTION)

MODEL INSTANCE



Learning

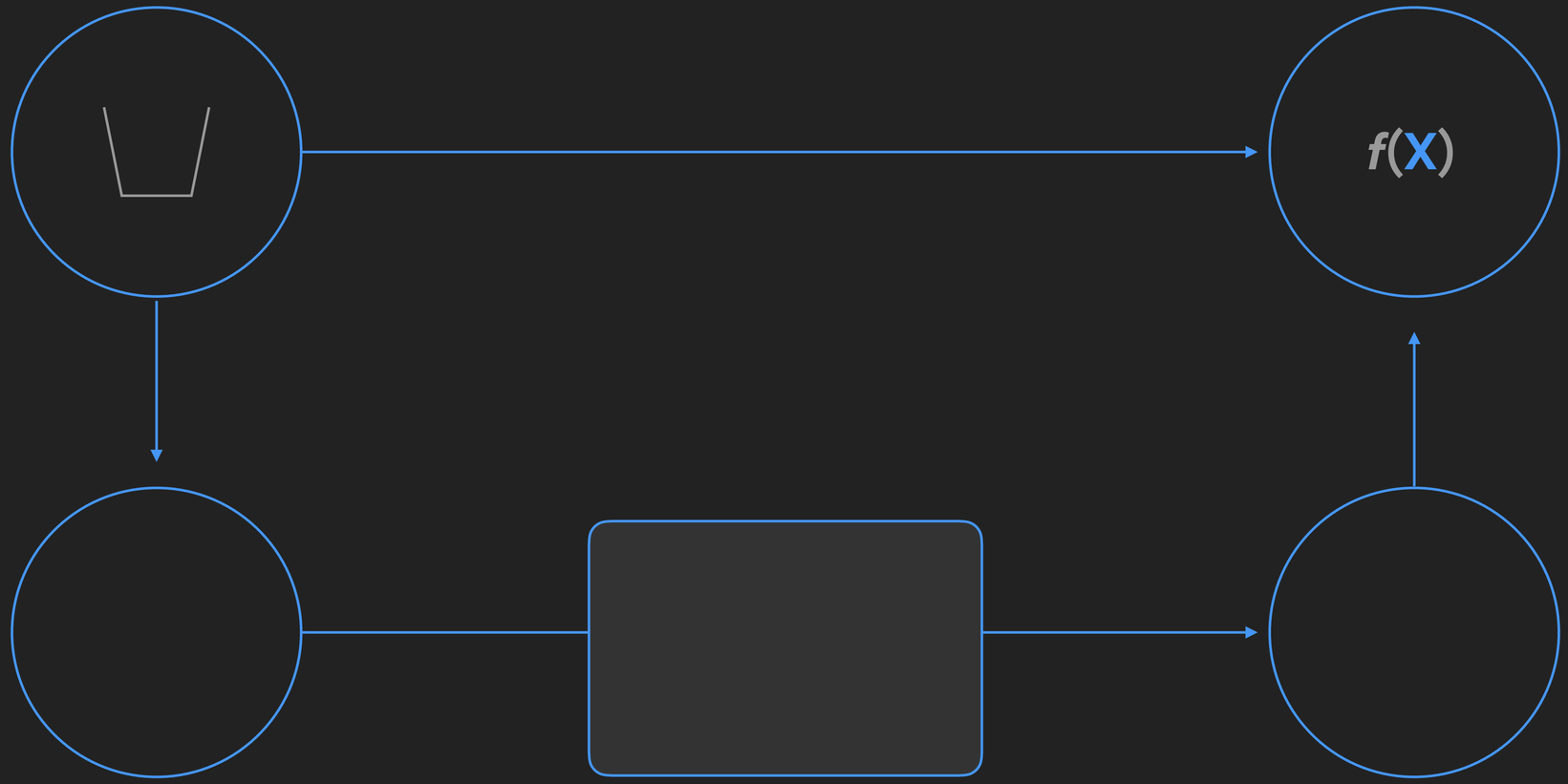


TRAINING DATA

BUCKET OF MODELS

NORMS (COST FUNCTION)

MODEL INSTANCE



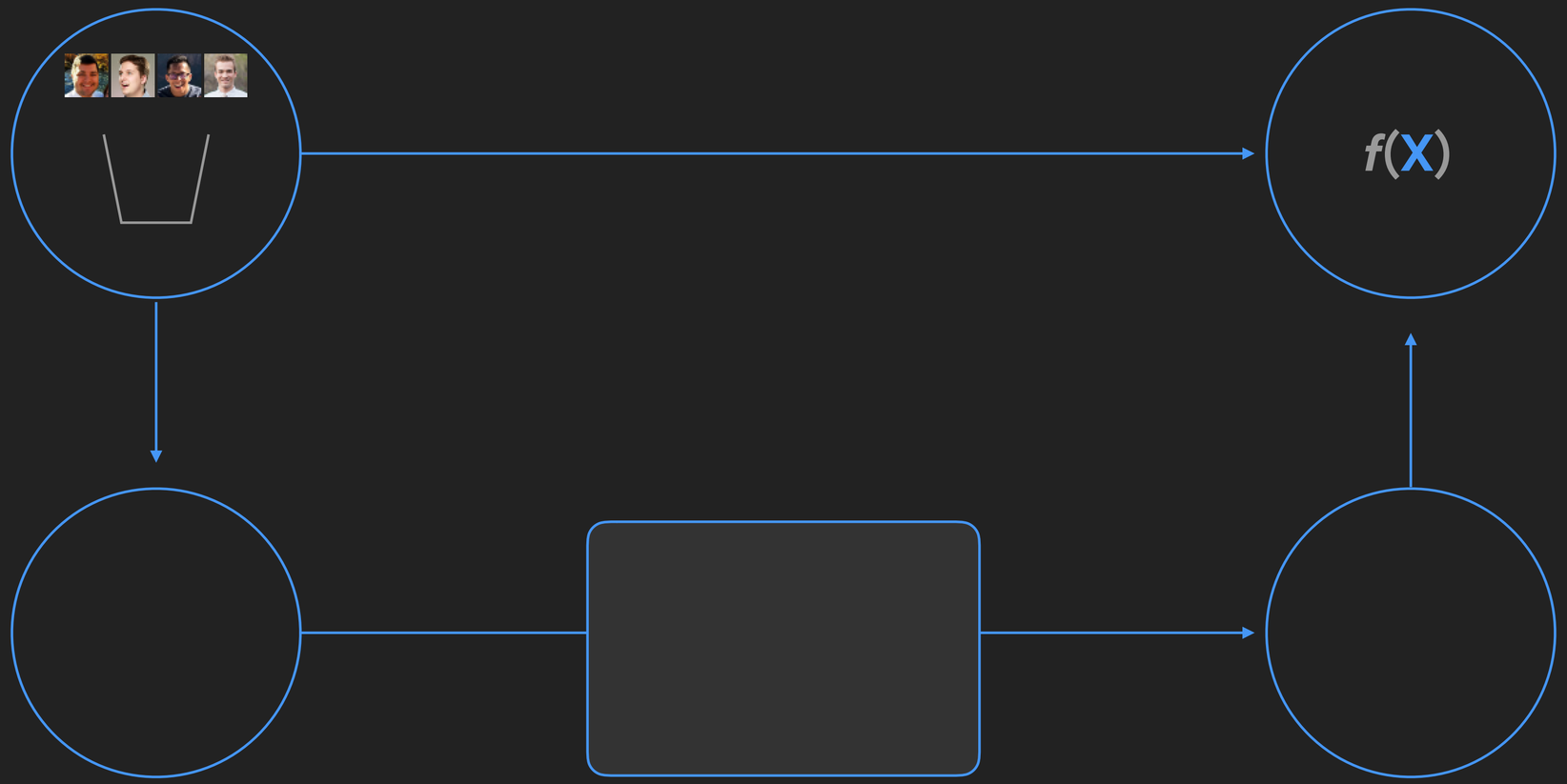
Learning

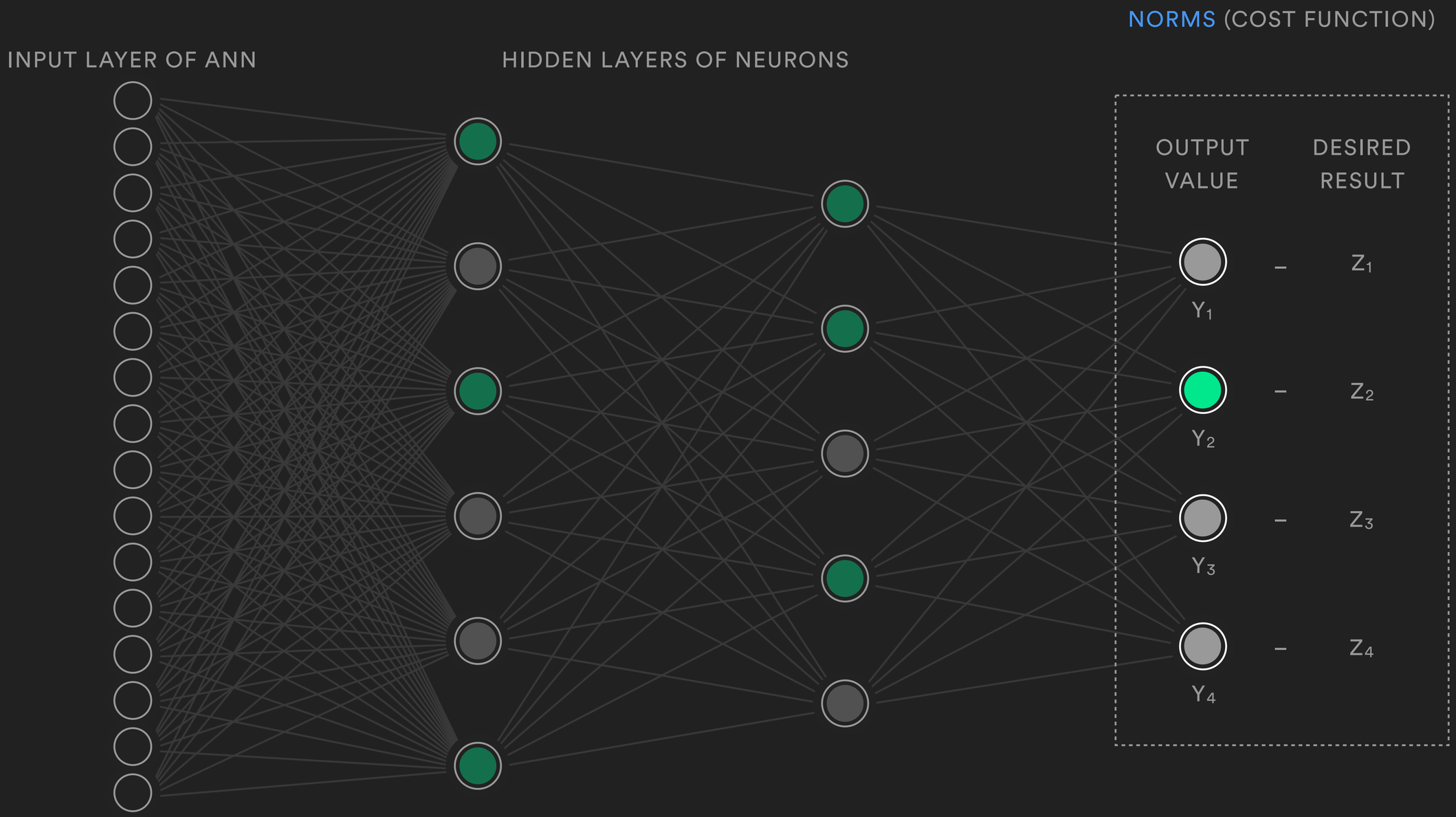
TRAINING DATA

BUCKET OF MODELS

NORMS (COST FUNCTION)

MODEL INSTANCE




NORMS (COST FUNCTION)
INPUT LAYER OF ANN
HIDDEN LAYERS OF NEURONS
OUTPUT VALUE
DESIRED RESULT
Y1
Z1
Y2
Z2
Y3
Z3
Y4
Z4

Cost Function

COST FUNCTION

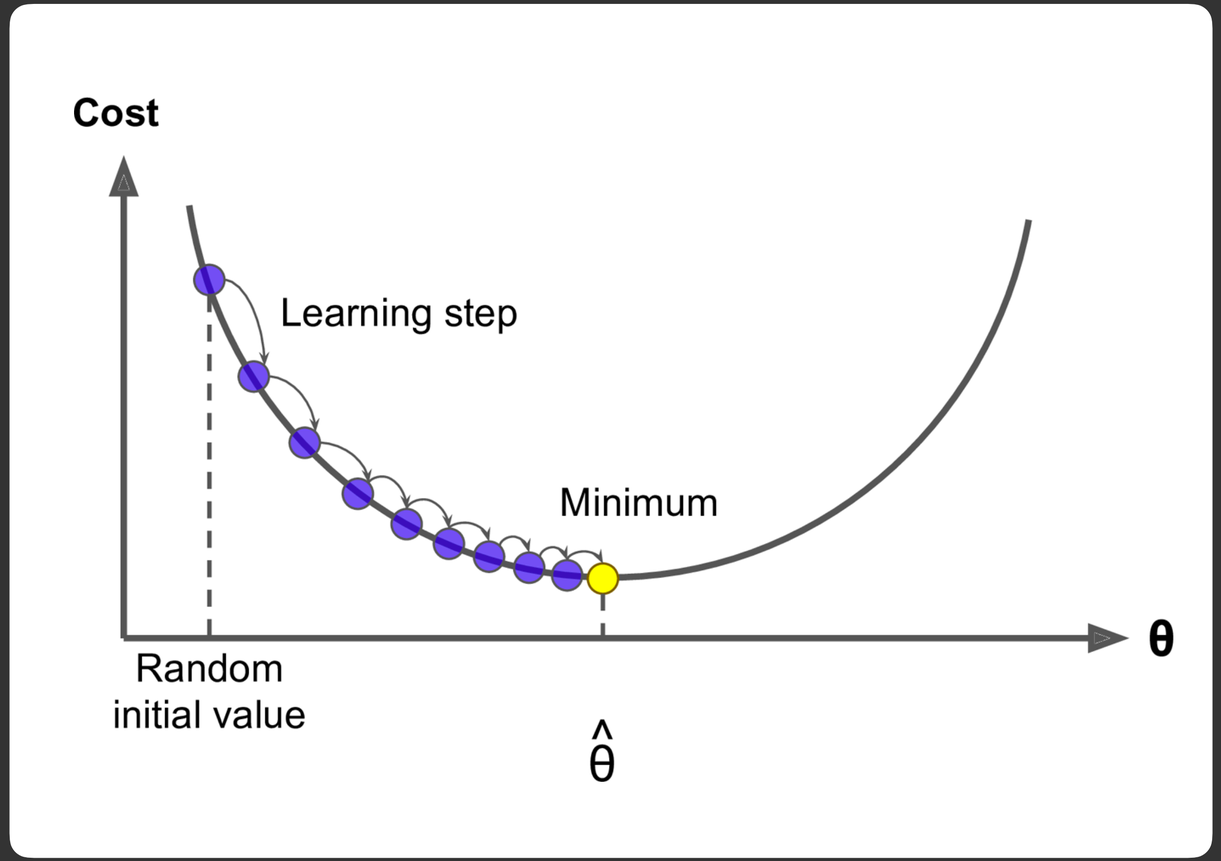
Looks at the outputs generated by the neural net and compares it to what the results *should* be

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

A neural net that produces an answer close to the desired answer will have a lower cost

GOAL

Find which weights and biases will minimize the cost function output, known as gradient descent



Learning

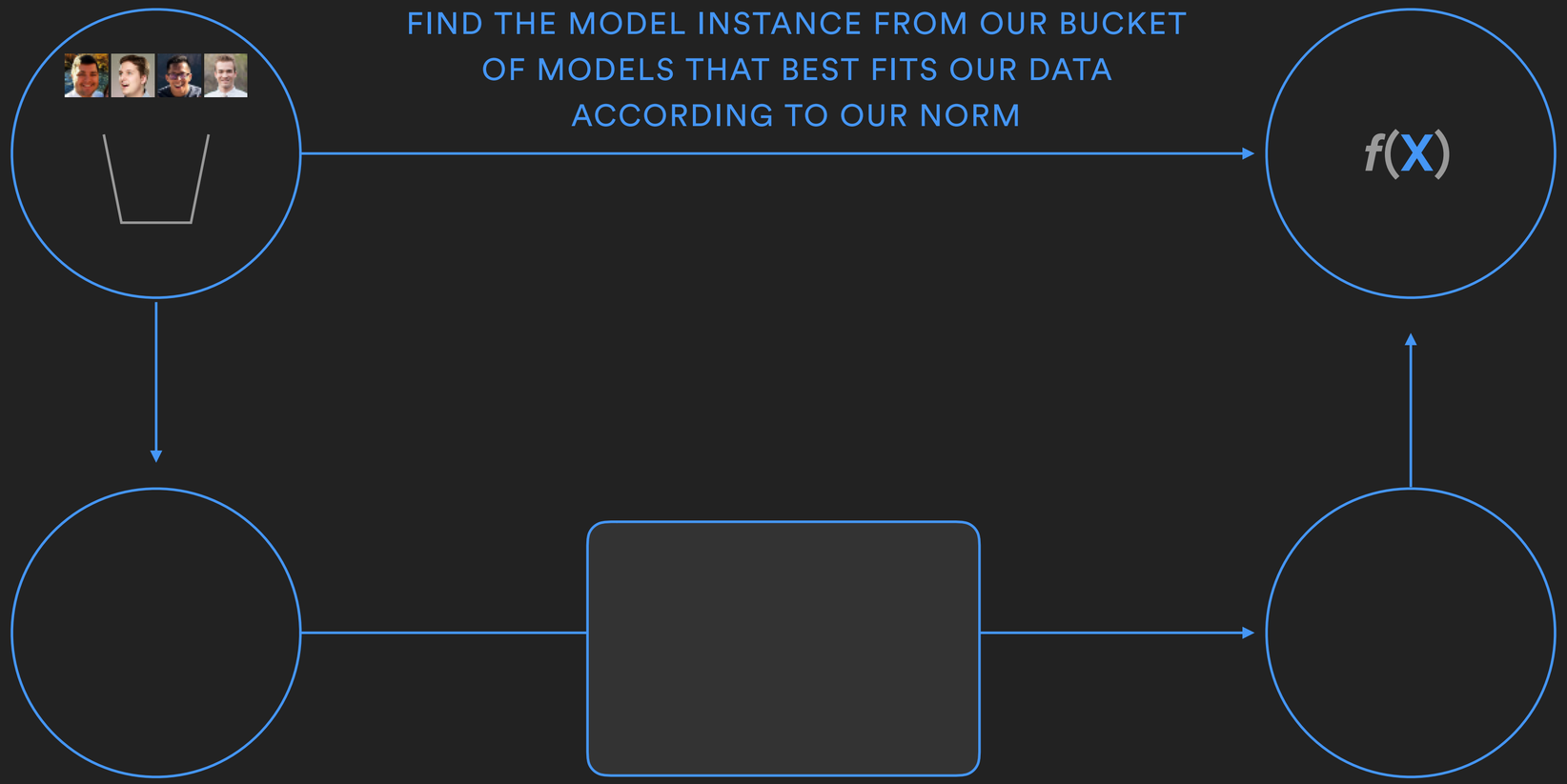
TRAINING DATA

BUCKET OF MODELS

NORMS (COST FUNCTION)

MODEL INSTANCE

FIND THE MODEL INSTANCE FROM OUR BUCKET OF MODELS THAT BEST FITS OUR DATA ACCORDING TO OUR NORM

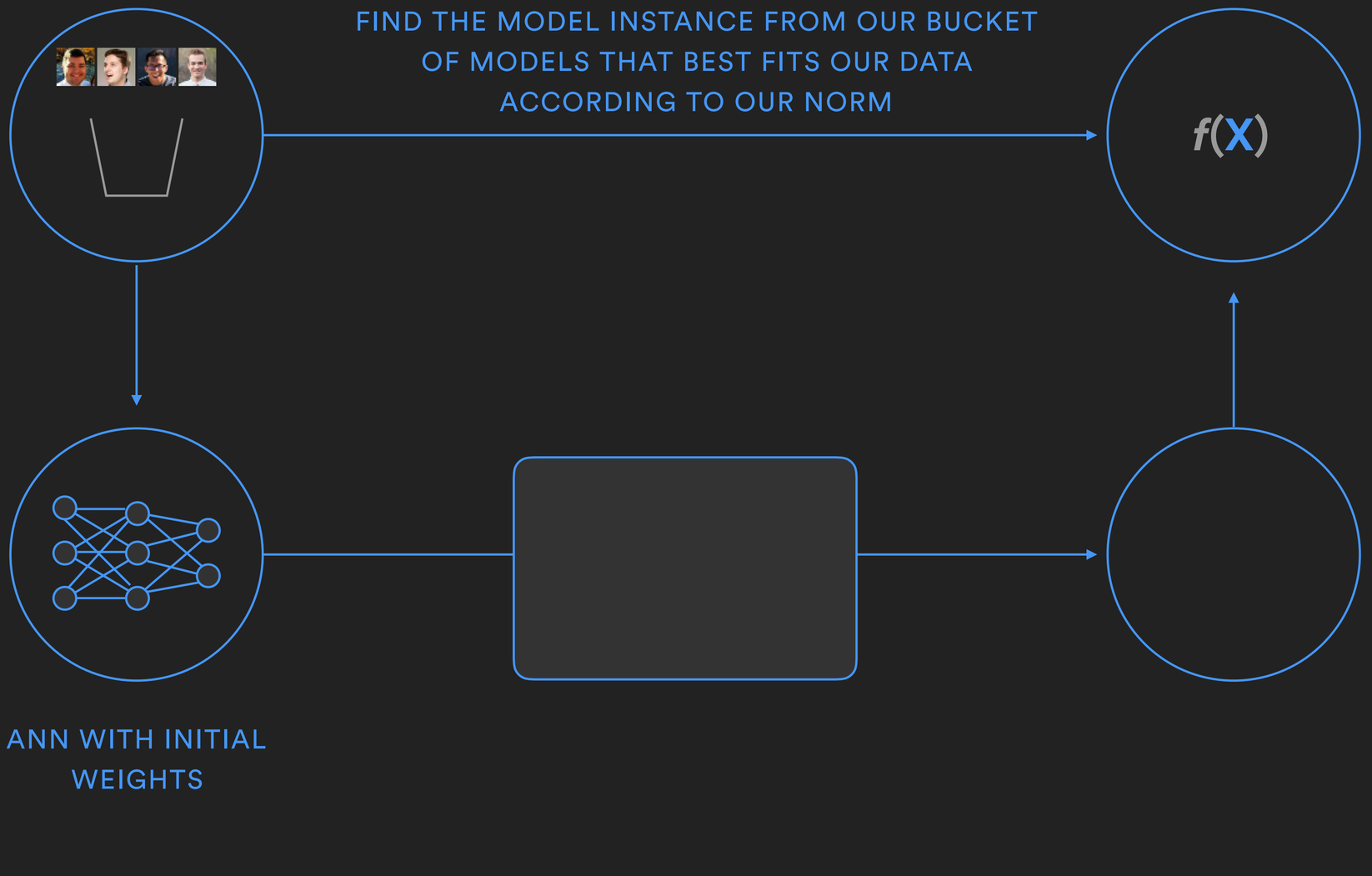


Learning

TRAINING DATA

BUCKET OF MODELS

NORMS (COST FUNCTION)



Learning

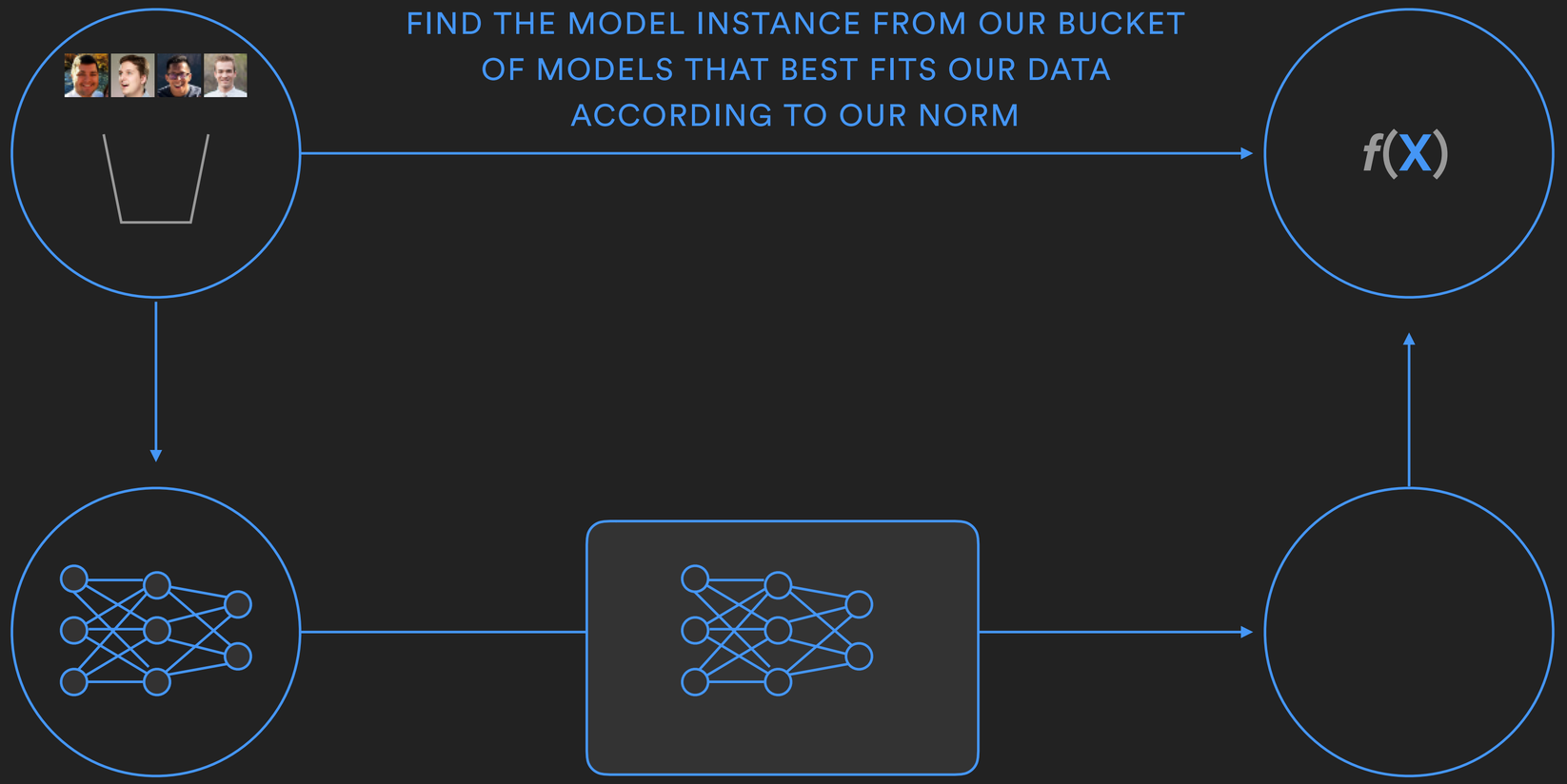
TRAINING DATA

BUCKET OF MODELS

NORMS (COST FUNCTION)

MODEL INSTANCE

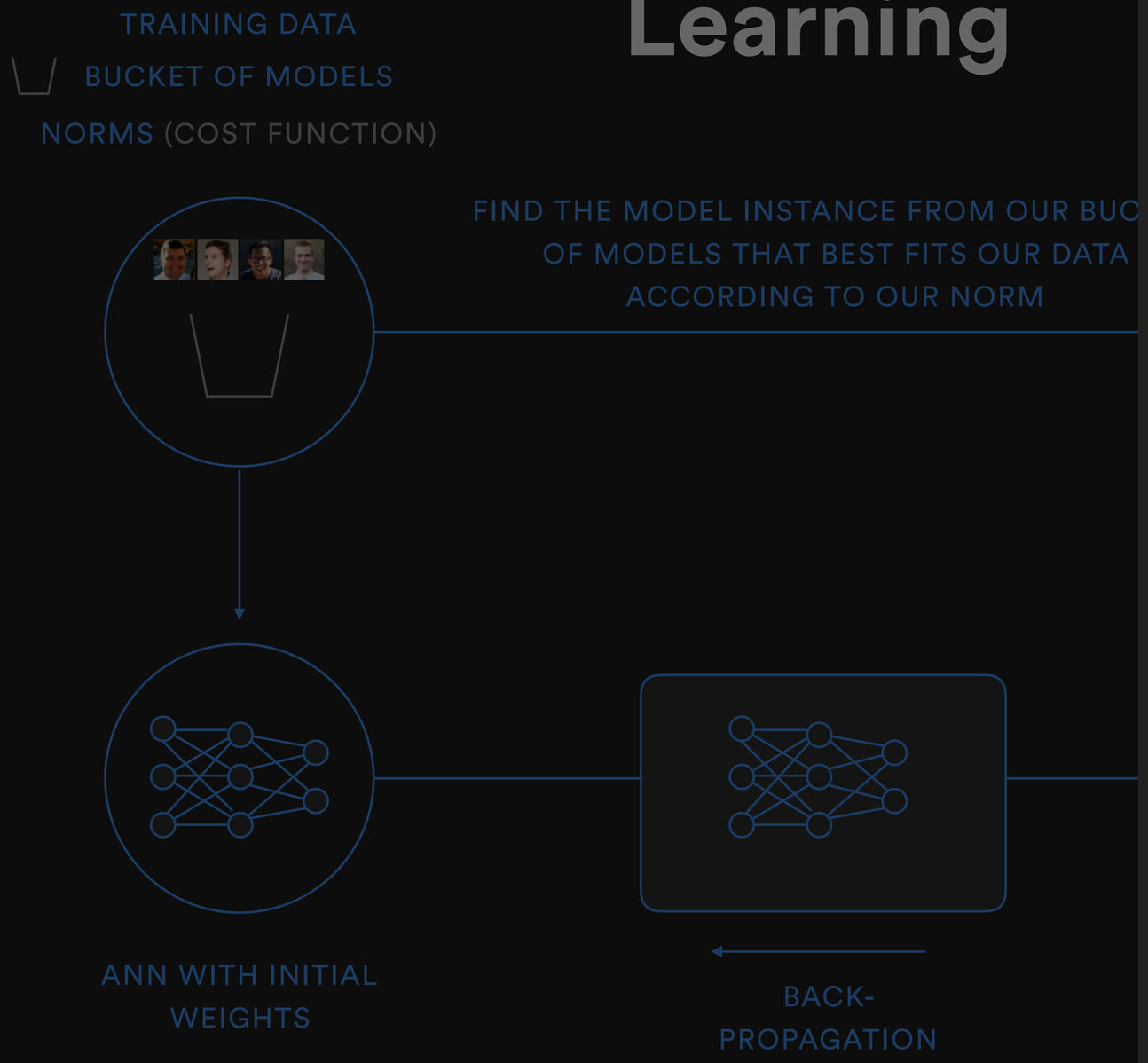
FIND THE MODEL INSTANCE FROM OUR BUCKET OF MODELS THAT BEST FITS OUR DATA ACCORDING TO OUR NORM

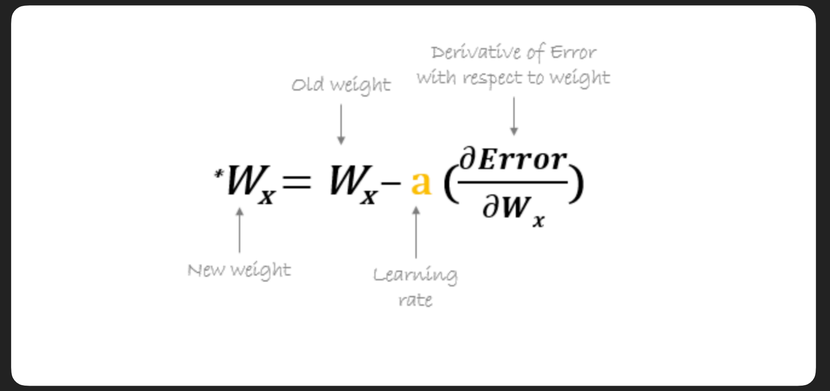


ANN WITH INITIAL WEIGHTS

BACK-PROPAGATION

Learning

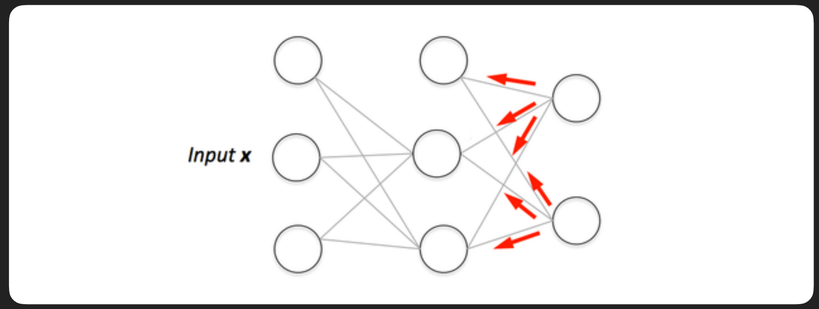




Sums up the total cost of each output:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Computes partial derivatives of of the total cost function with respect to any weight *w* or bias *b* in the network.



Applies small changes to the weights and biases in each layer.

Automatically adjusts the weights between each set of data. No manual tuning required

Learning

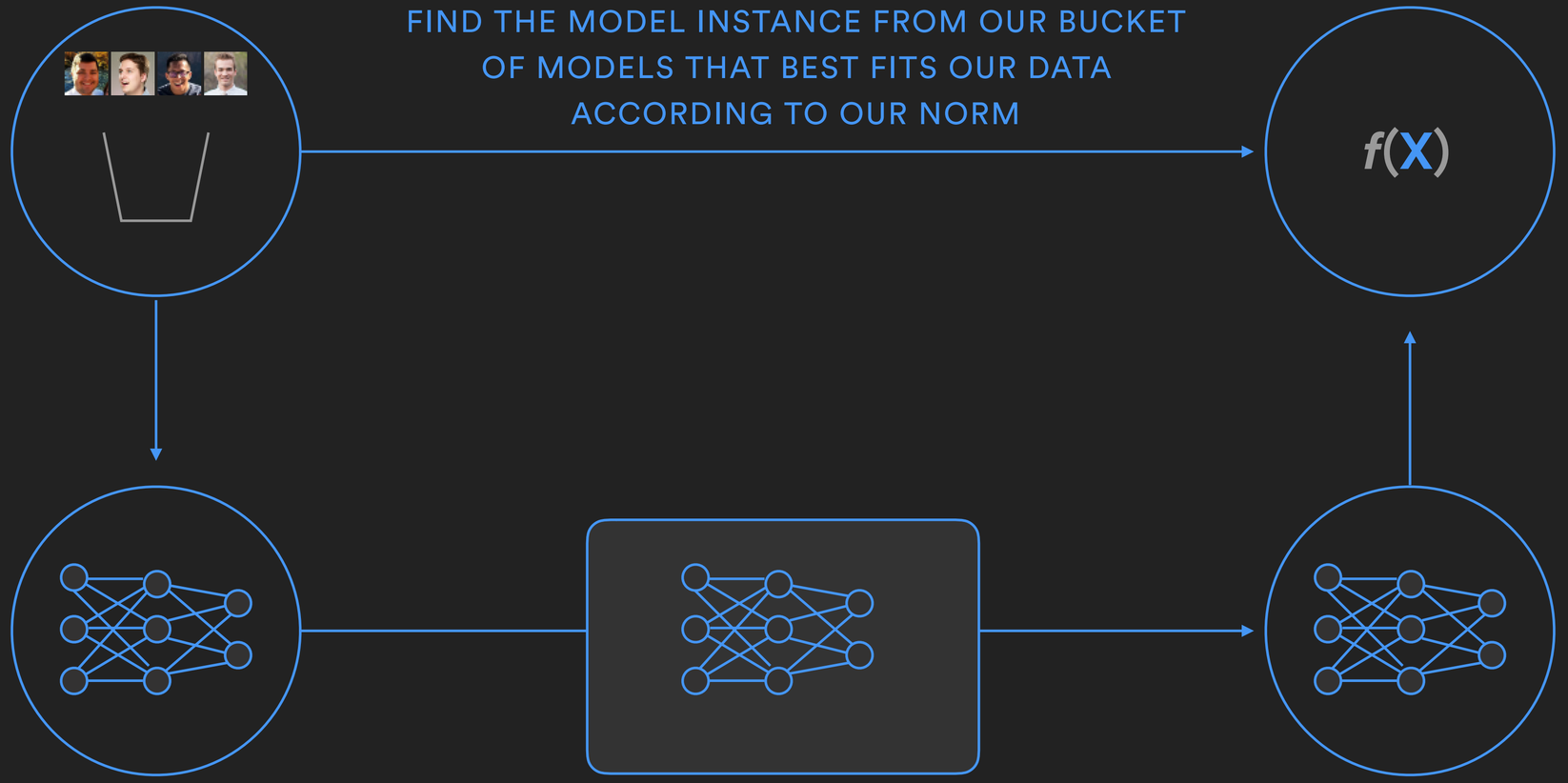
TRAINING DATA

BUCKET OF MODELS

NORMS (COST FUNCTION)

MODEL INSTANCE

FIND THE MODEL INSTANCE FROM OUR BUCKET OF MODELS THAT BEST FITS OUR DATA ACCORDING TO OUR NORM



ANN WITH INITIAL WEIGHTS

BACK-PROPAGATION

ANN WITH TUNED WEIGHTS

Using Evolutionary Algorithms in ANNs

STEP 1

Generate population of *n* random initial sets of weights and biases.
Known as *species.*

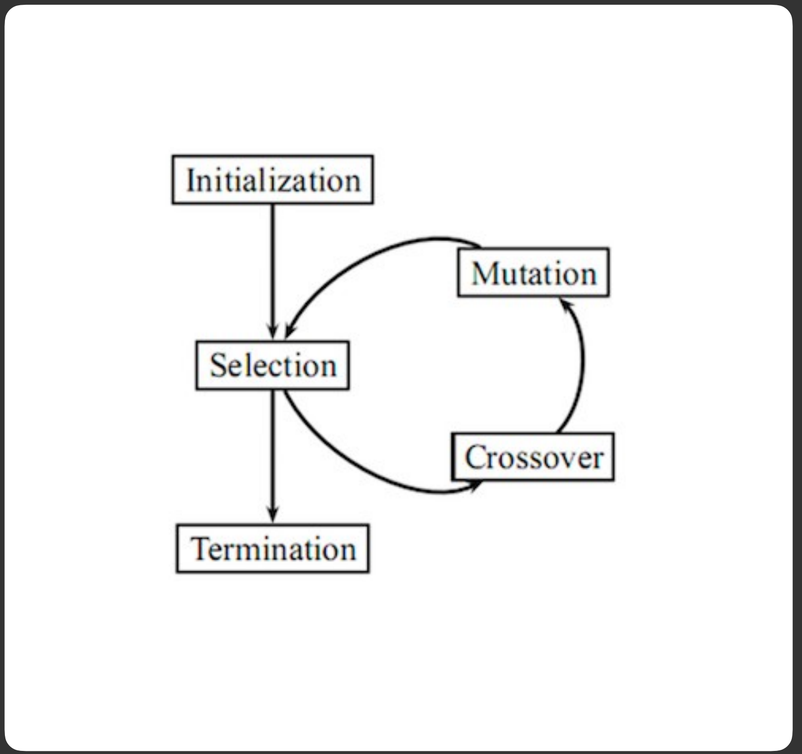
STEP 2

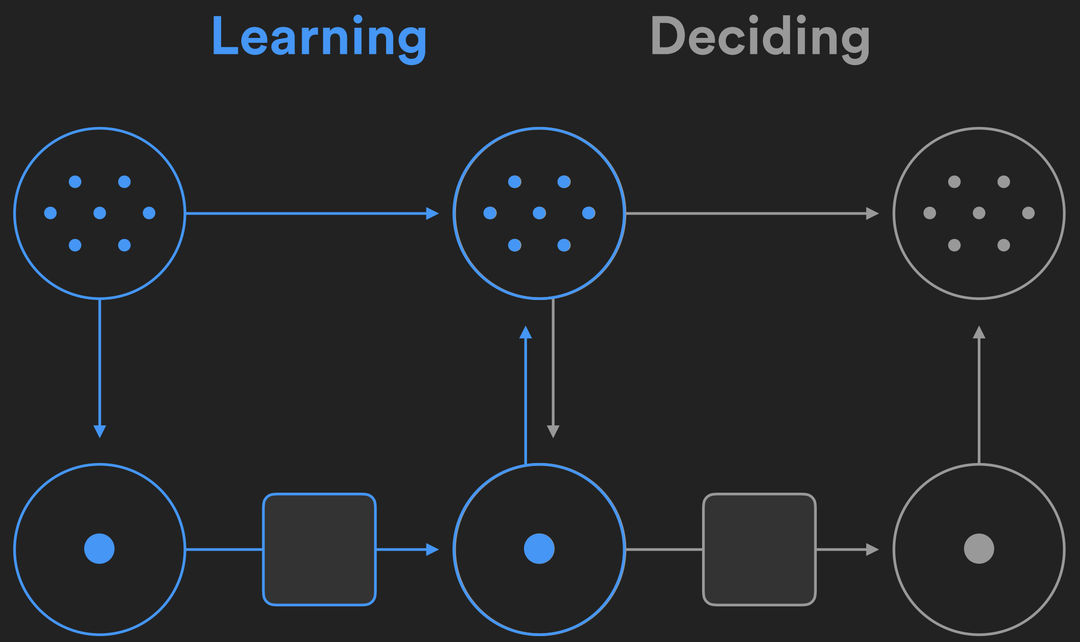
Evaluate the fitness of each species in that population
(time limit, lowest cost achieved, etc.)

STEP 3 - REPEAT THE FOLLOWING

a. Select the best-fit individuals for reproduction. (Parents)
b. Breed a new generation of new species
   through crossover and mutation.
c. Evaluate the individual fitness of new individuals.
d. Replace least-fit population with new individuals.

Over time, this process selects the best set of weights and biases it can, finding a local minimum of the function and reducing error



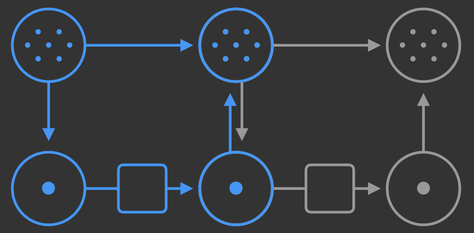

Learning
Deciding

INTRODUCTION

DECIDING

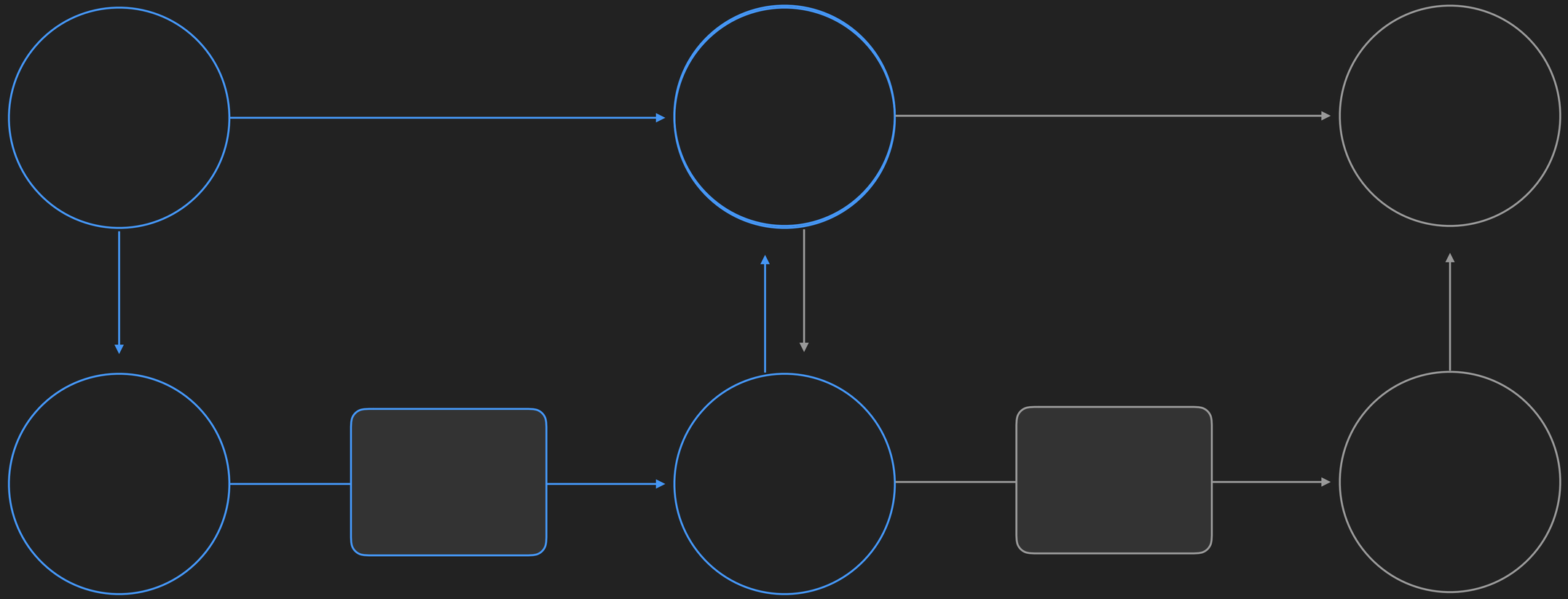
LEARNING

CORRECTNESS



Learning

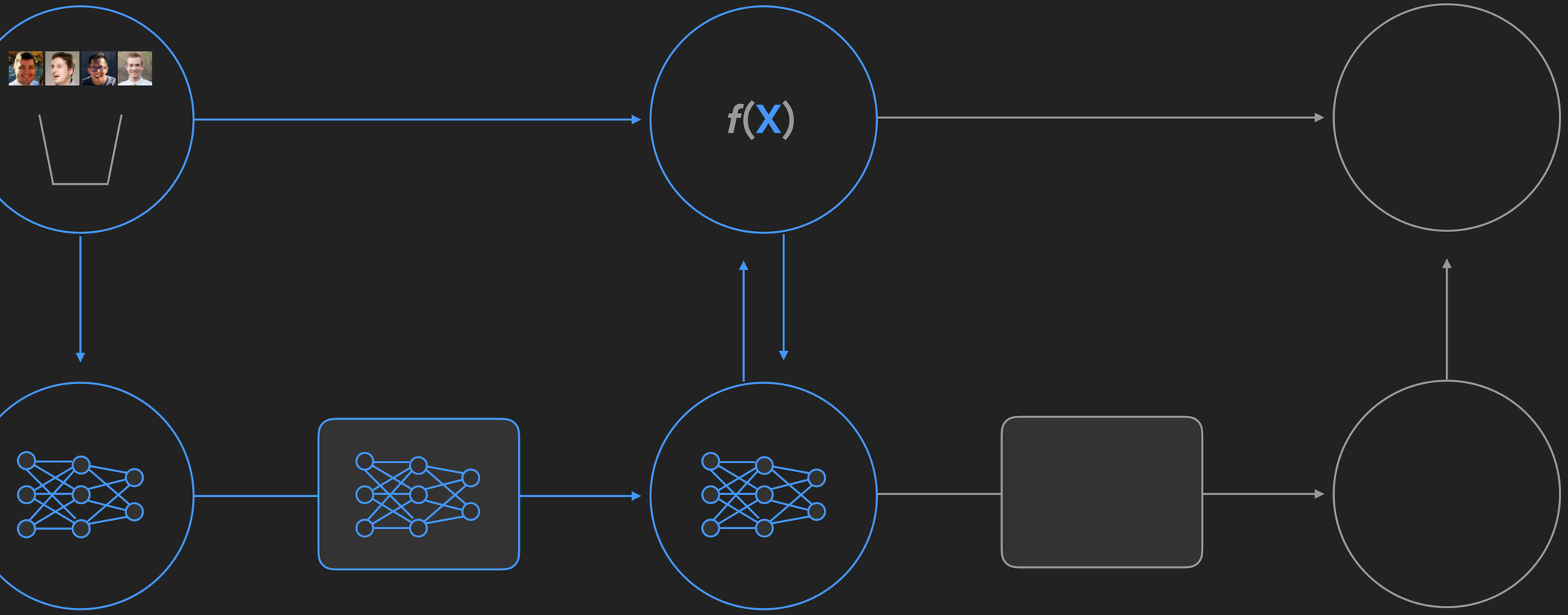
Deciding



TRAINING DATA

BUCKET OF MODELS

NORMS (COST FUNCTION)



ANN WITH INITIAL WEIGHTS

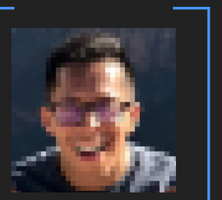
BACK-PROPAGATION

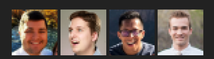
ANN WITH TUNED WEIGHTS

TRAINING DATA

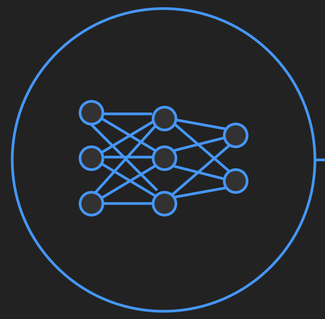
BUCKET OF MODELS

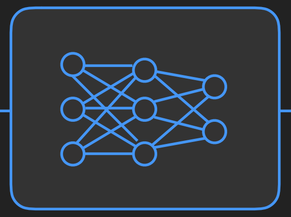
NORMS (COST FUNCTION)

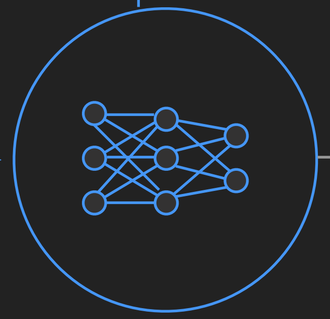


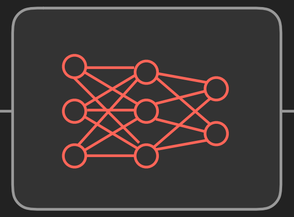


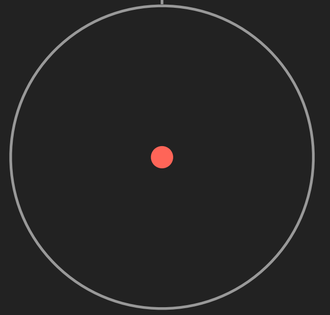
f(**X**)











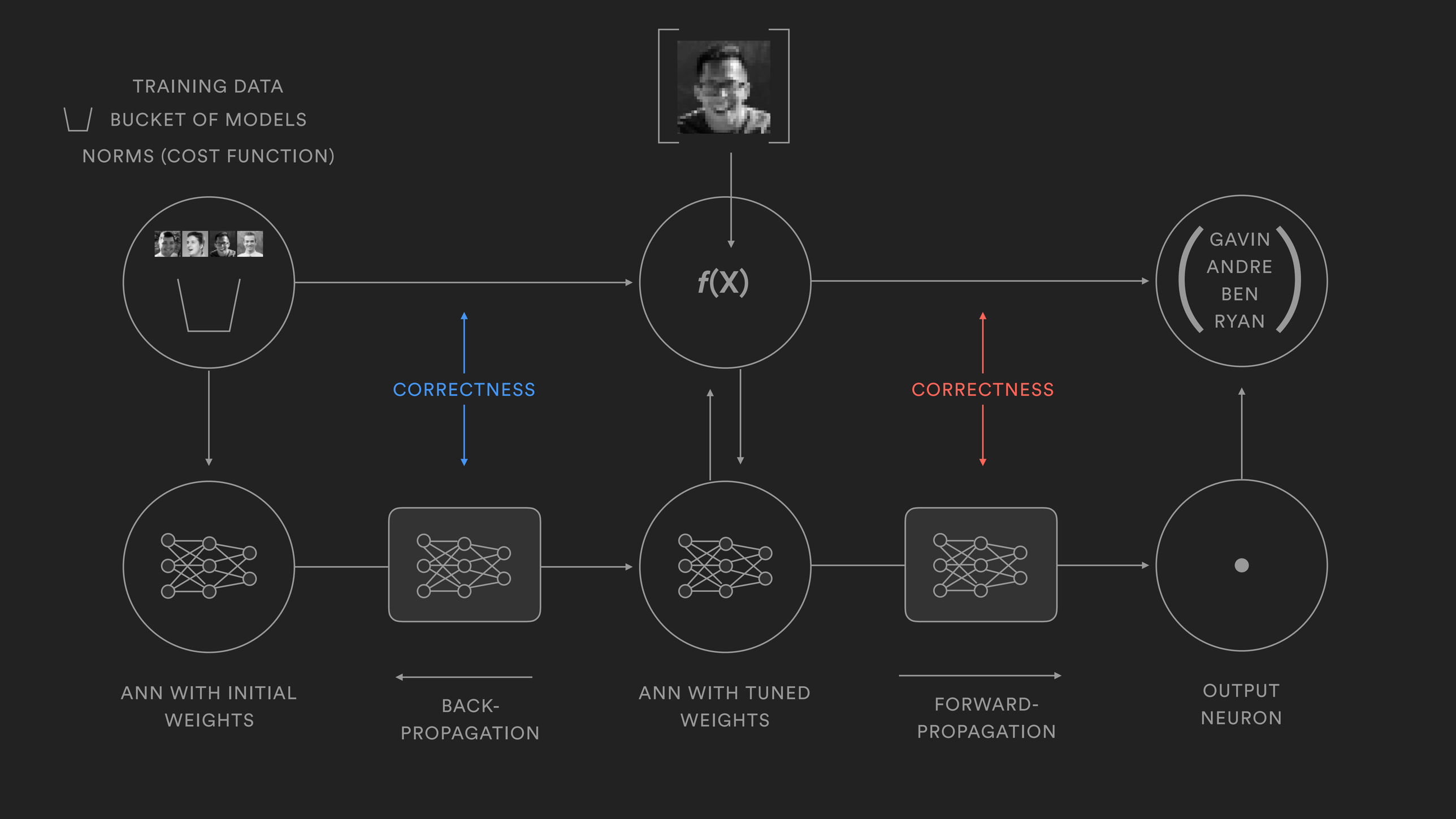
ANN WITH INITIAL WEIGHTS

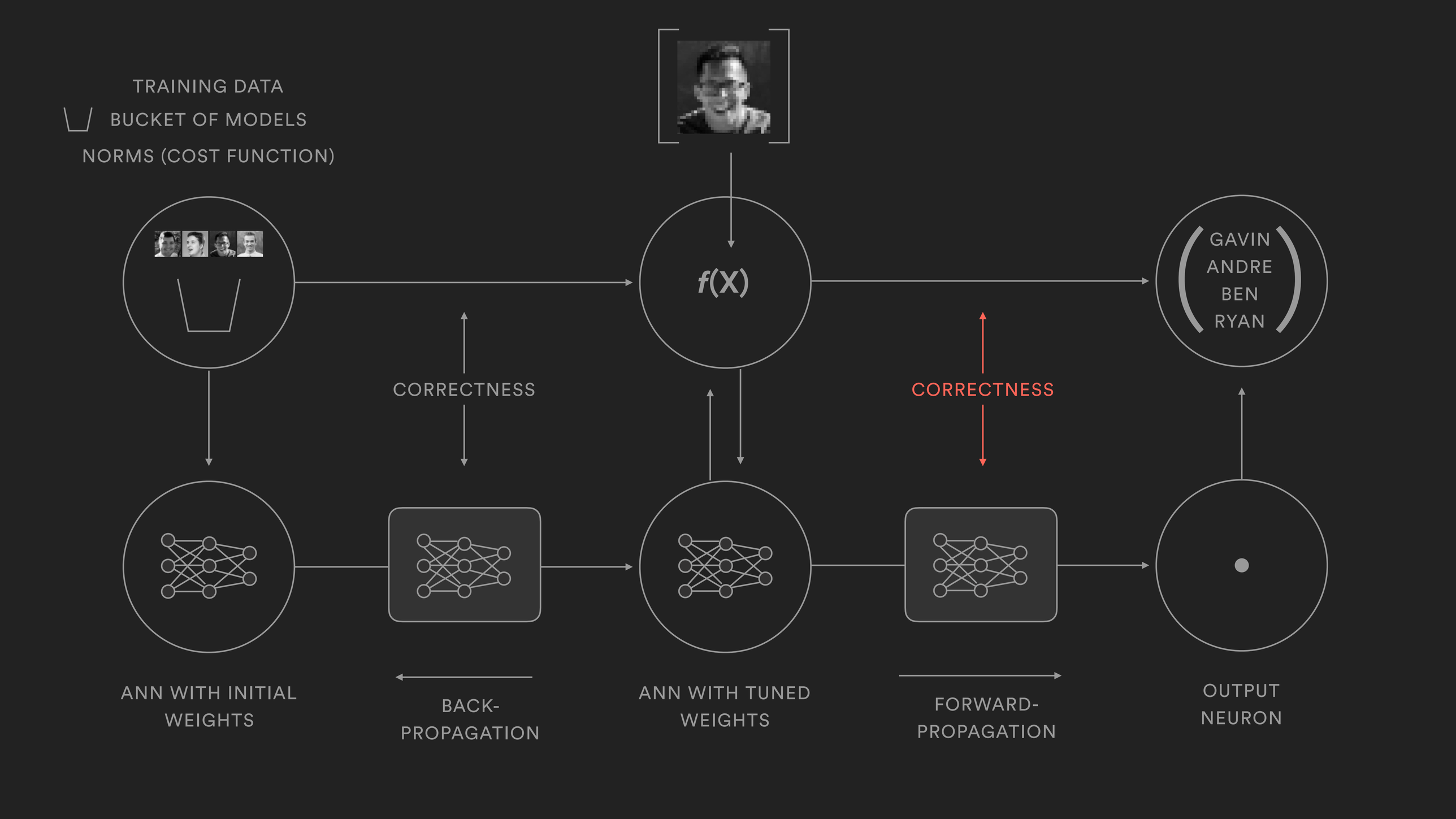
BACK-PROPAGATION

ANN WITH TUNED WEIGHTS

FORWARD-PROPAGATION

OUTPUT NEURON


TRAINING DATA
BUCKET OF MODELS
NORMS (COST FUNCTION)
f(X)
GAVIN
ANDRE
BEN
RYAN
CORRECTNESS
CORRECTNESS
ANN WITH INITIAL WEIGHTS
BACK-PROPAGATION
ANN WITH TUNED WEIGHTS
FORWARD-PROPAGATION
OUTPUT NEURON

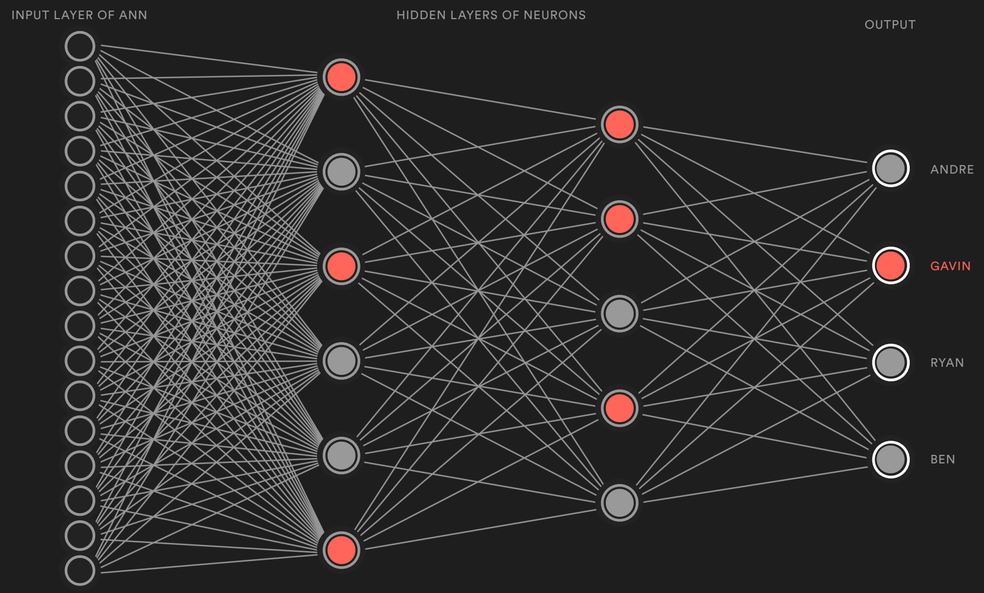

TRAINING DATA
BUCKET OF MODELS
NORMS (COST FUNCTION)
f(X)
GAVIN
ANDRE
BEN
RYAN
CORRECTNESS
CORRECTNESS
ANN WITH INITIAL WEIGHTS
BACK-PROPAGATION
ANN WITH TUNED WEIGHTS
FORWARD-PROPAGATION
OUTPUT NEURON

From this a skeleton can be extracted

From this a skeleton can be extracted

Universal approximation theorem

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbf{R}_n$, under mild assumptions on the activation function.



The theorem in mathematical terms:

Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the *m*-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

as an approximate realization of the function f where f is independent of φ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

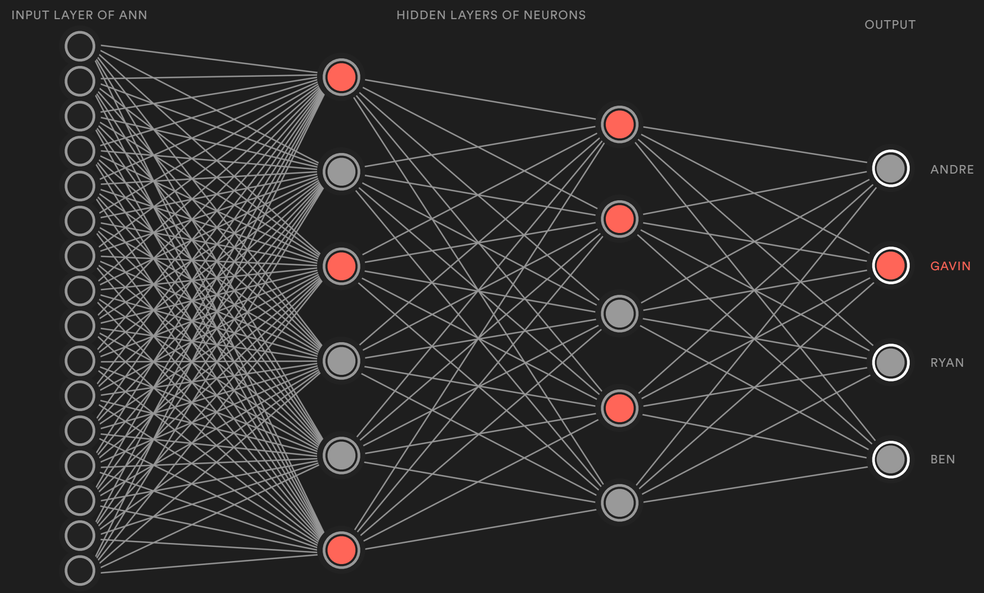
Universal approximation theorem

UAT AND ANN CORRECTNESS SIMPLIFIED

Any continuous function can be approximated by an ANN with a finite number of neurons

WHAT WE LEARN FROM ANN CORRECTNESS

A correct result can’t be guaranteed
A ‘Good’ result is guaranteed to be *possible*

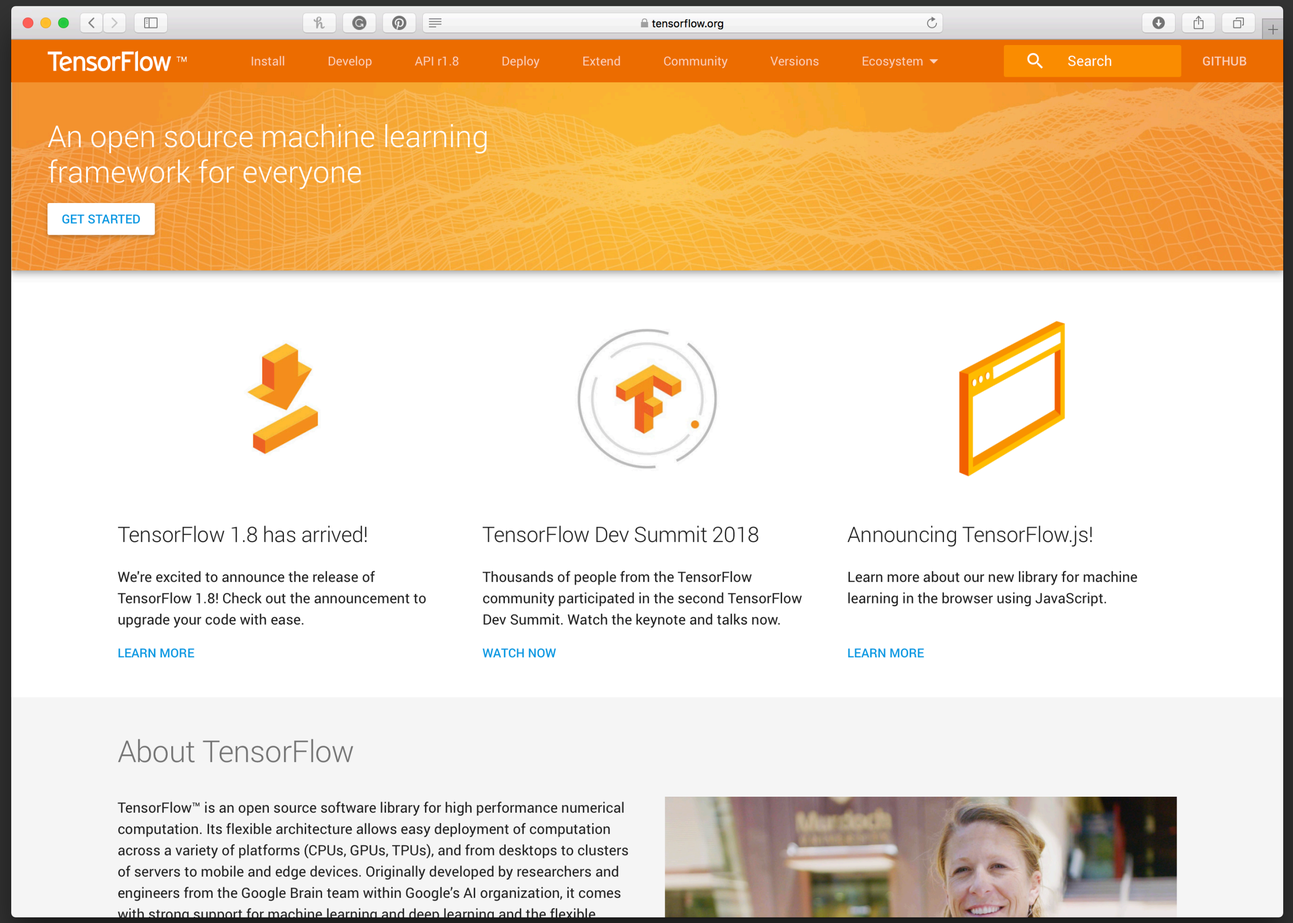


“A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.”

— Ian Goodfellow

Resources

- Tensor Flow
- Pytorch
- MXNet
- Caffe
- Theano
- Amazon Machine Learning



APPENDIX

"Little more than speculation and wishful thinking ties the actual work in AI to the mysterious workings of the human mind."

—Jerry Kaplan, an AI professor at Stanford University

Comparison between ANNs and Brains

TOO NEAT

Human-built networks prioritize mathematical elegance and power. Brains do not.

TOO SIMPLE

One neuron in the brain is as complex as a supercomputer

TOO FEW

Number of neurons in the brain is roughly equivalent to number of stars in the galaxy (about 85 Billion) Each Neuron is wired to 1000 others. About 85 trillion connections synapses.

TOO DRY

ANN researchers typically ignore biophysics

Biological Neuron Network

Purely feedforward. Brains contain feedback connections in various directions, but each on his strictly one-way

Number of neurons in the brain is roughly equivalent to number of stars in the galaxy (about 100 Billion)

Each Neuron is wired to 1000 others. About 100 trillion connections (synapses)

Artificial Neuron Network

Can apply back-propagation

Top-down organization

Where the computational procedure is constructed according to some well-defined and clearly understood computational procedure, where this procedure provides a clear cut solution to some problem at hand. (Euclid's Algorithm)

AKA GOFAI

Bottom-up organization

Clearly defined rules are not specified in advance, but instead there is a procedure laid down for the way that the system is to 'learn' and to improve its performance according to its 'experience". Thus the rules are subject to continual modification. (Comparing results of algorithm to correct answers as in ANNs)

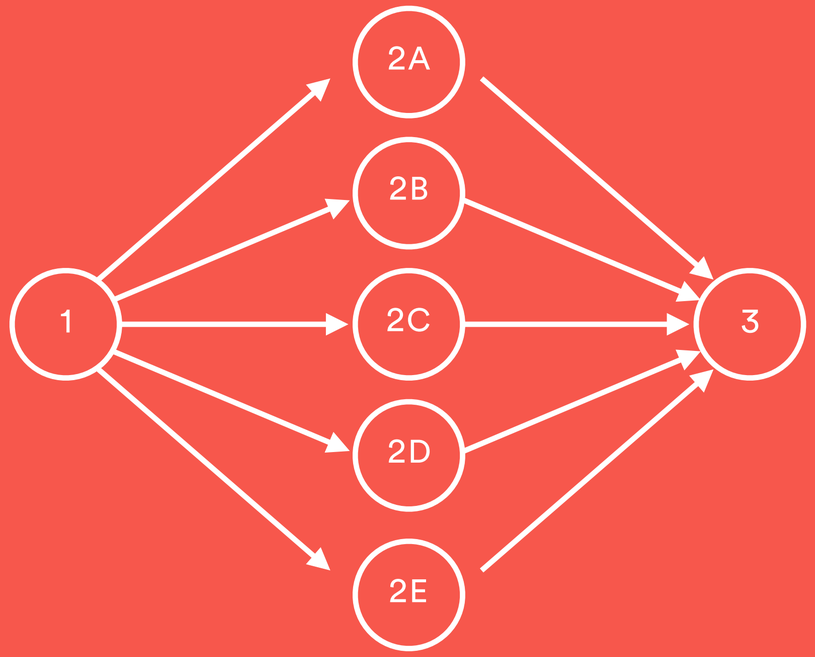
NOUVELLE AI

Serial Architecture

STEP-BY STEP

Parallel Architecture

DOES MANY INDEPENDENT
COMPUTATIONS SIMULTANEOUSLY.



Universal approximation theorem

For every function there exists an ANN with some hidden layer that approximates that function.

Assumes that you can have an infinite number of neurons.

We know of no way of determining the Model of the hidden layers in the neural networks.