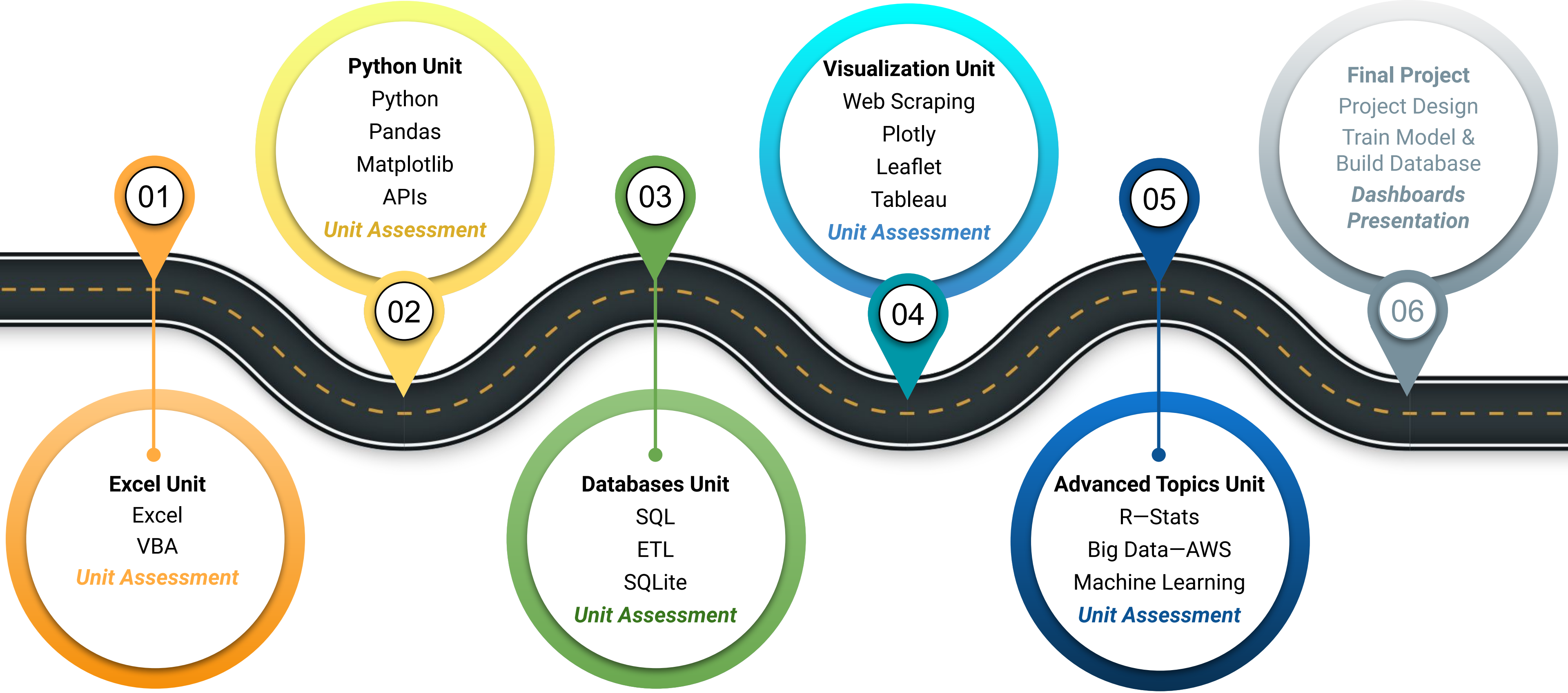
Introduction to Advanced Machine Learning

Data Boot Camp

Lesson 19.1

The Big Picture



Pro Tip:

We’re getting close to our end-of-course capstone project! Collaboration will be key, so be ready to dive in soon!

Module 19

This Week: Advanced Machine Learning

This Week: Advanced Machine Learning

By the end of this week, you'll know how to:

- Describe the perceptron model and its components
- Save and implement neural network models using TensorFlow
- Explain how different neural network structures change algorithm performance
- Preprocess and construct datasets for neural network models
- Implement deep neural network models using TensorFlow

This Week's Challenge

Using the skills learned throughout the week in machine learning and neural networks, you will use a dataset to create a binary classifier capable of predicting success for applicants seeking funding.

Career Connection

How will you use this module's content in your career?

Module 19

How to Succeed This Week

Pro Tip:

This week, we'll get into neural networks and deep learning!
There's a lot more to learn in these areas,
so keep up with your study habits!

Module 19

Today's Agenda

Today’s Agenda

By completing today’s activities, you’ll learn the following skills:

01 Comparing the traditional ML classification and regression models with neural network models

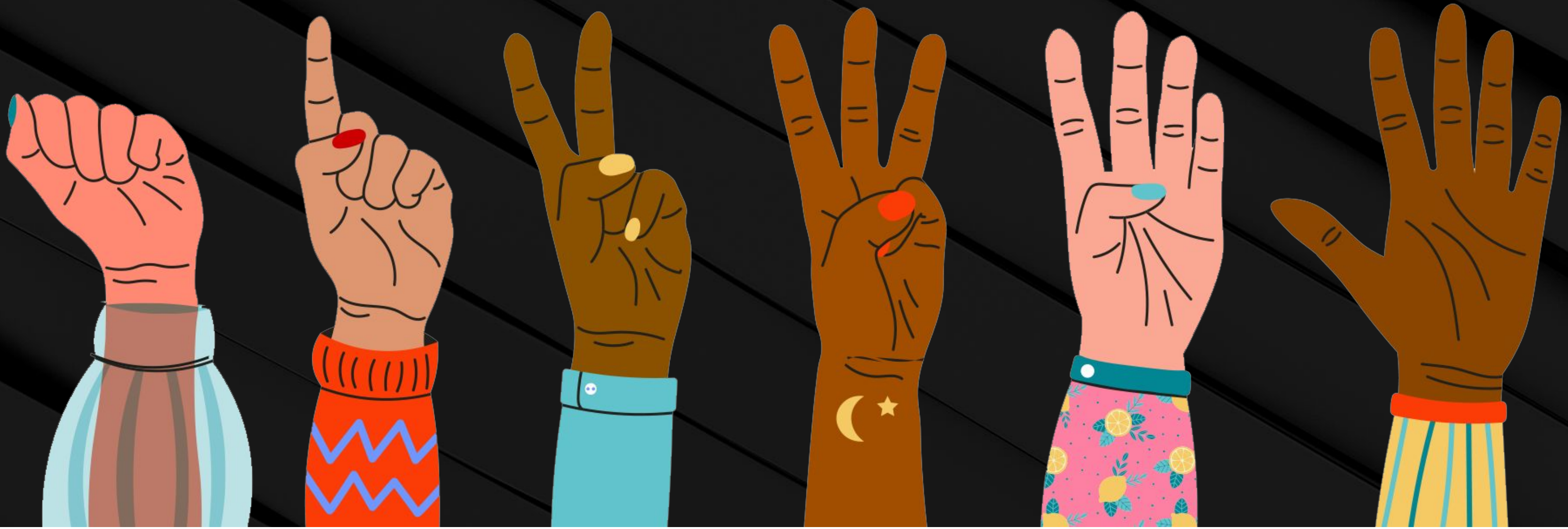
02 Describing and implementing a neural network model with TensorFlow

03 Running cloud-based neural network model Jupyter notebooks with Google Colab

Make sure you’ve downloaded any relevant class files!

FIST TO FIVE:

How comfortable do you feel with this topic?



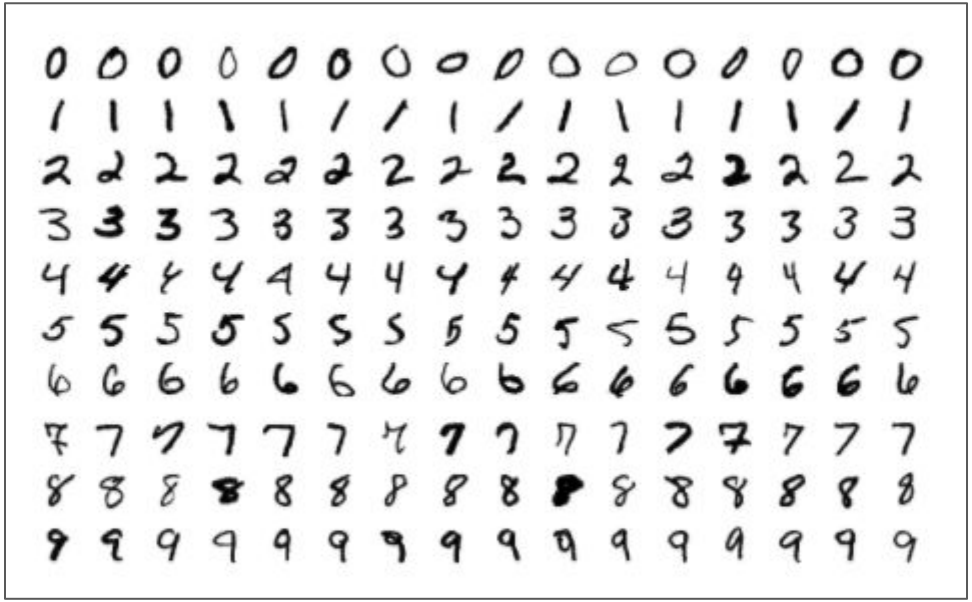
Introduction to Advanced Machine Learning

Surfing the Neural Net

Surfing the Neural Net

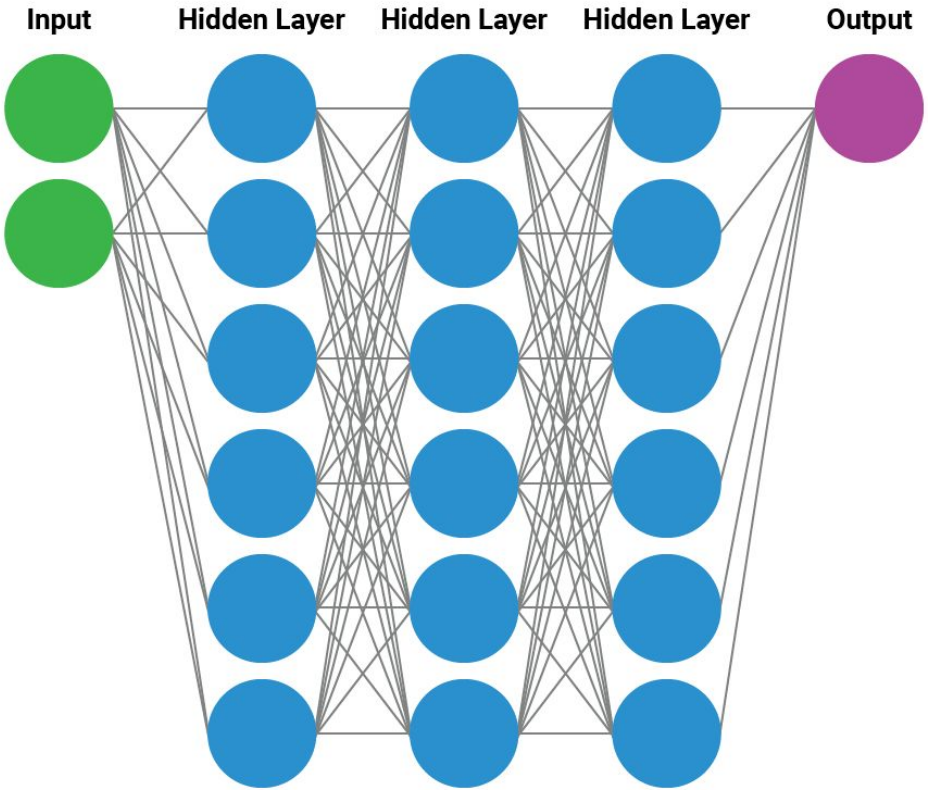
The **MNIST** database:

- The MNIST (Modified National Institute of Standards and Technology) dataset contains black-and-white images of handwritten numbers.
- A neural network can train on each pixel of each image as a scaled value from zero (completely white) to one (completely black). With enough data points, a trained neural network model can classify handwritten numbers with a high degree of accuracy.



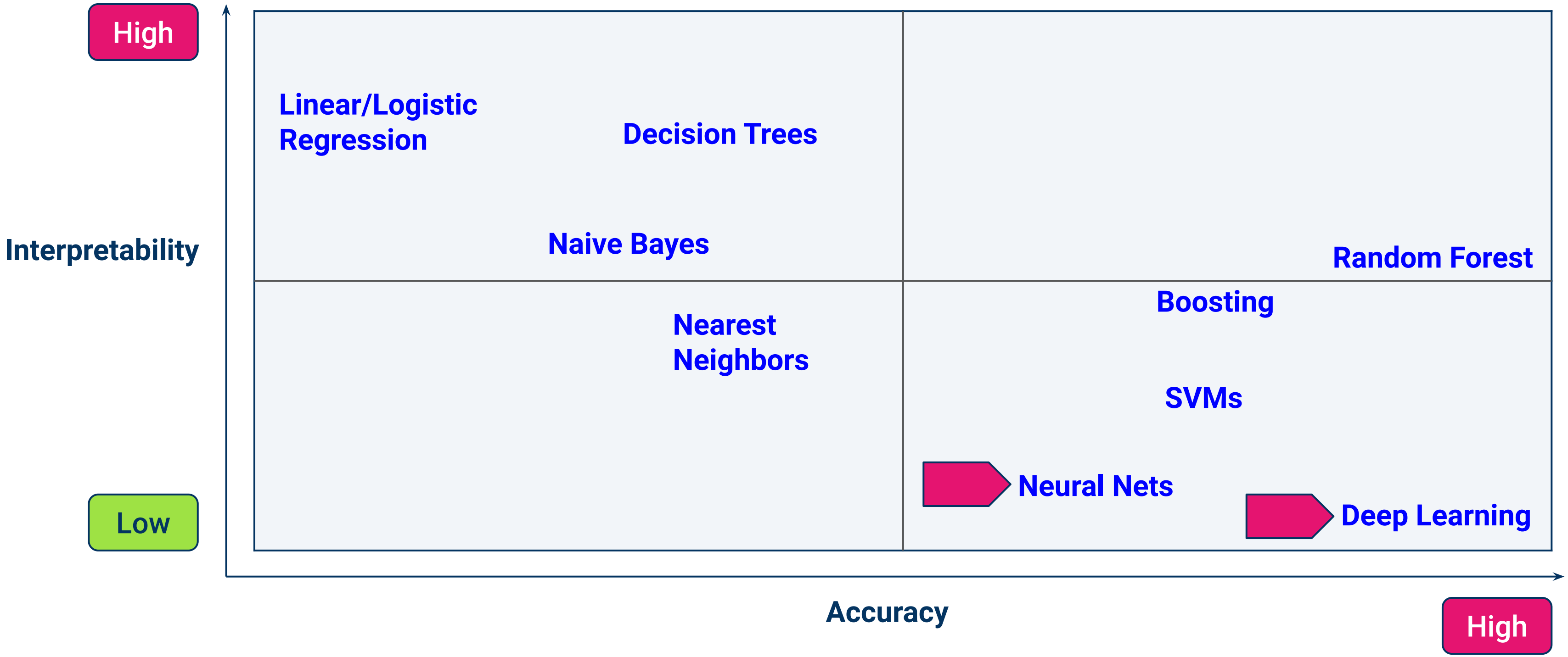
Sample images from MNIST test dataset

Example Diagram of a Neural Network



Surfing the Neural Net

Different types of machine learning:

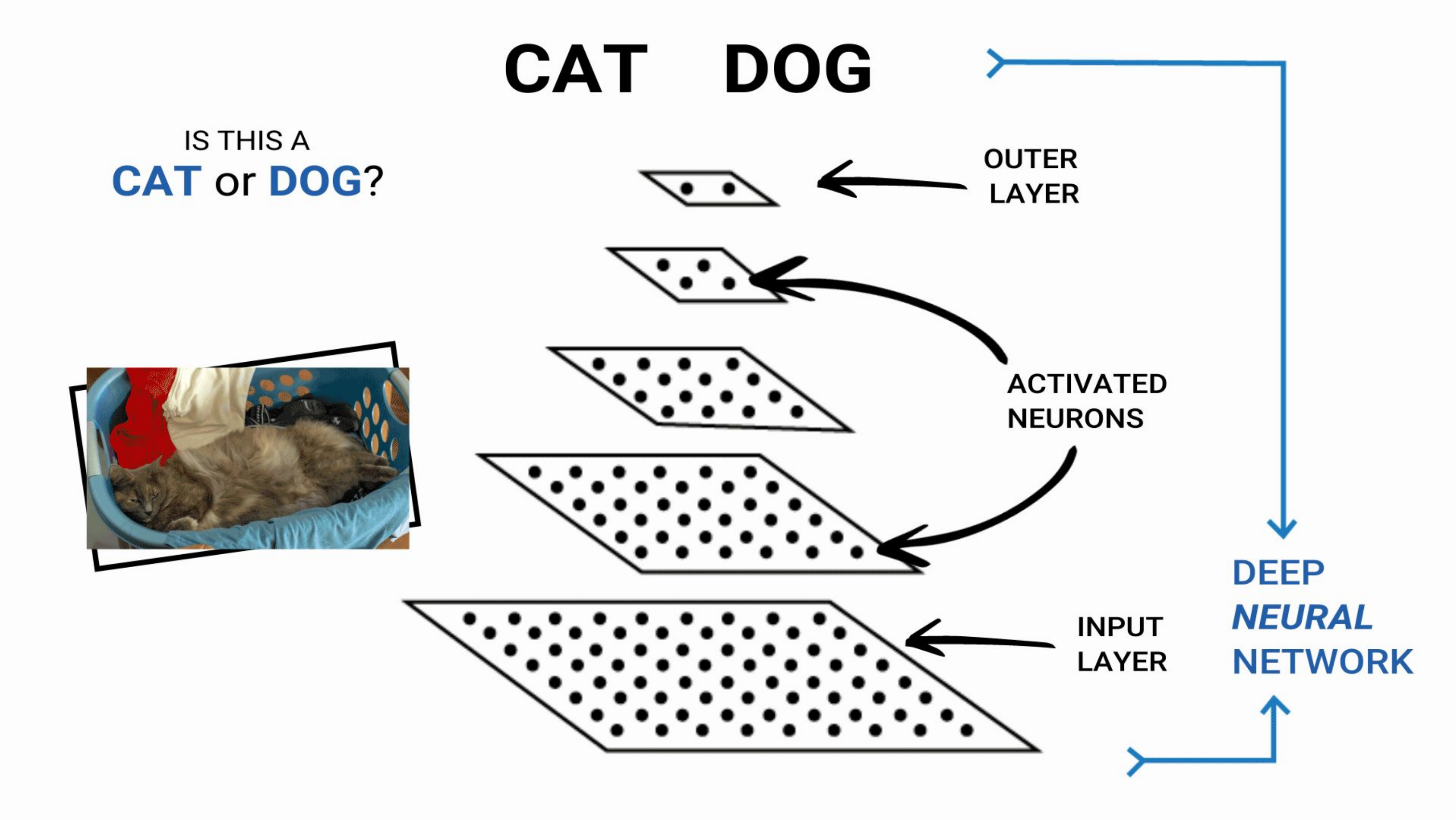


What Is a Neural Network?

Neural networks are an advanced form of machine learning that contain multiple layers of nodes, which perform individual computations.

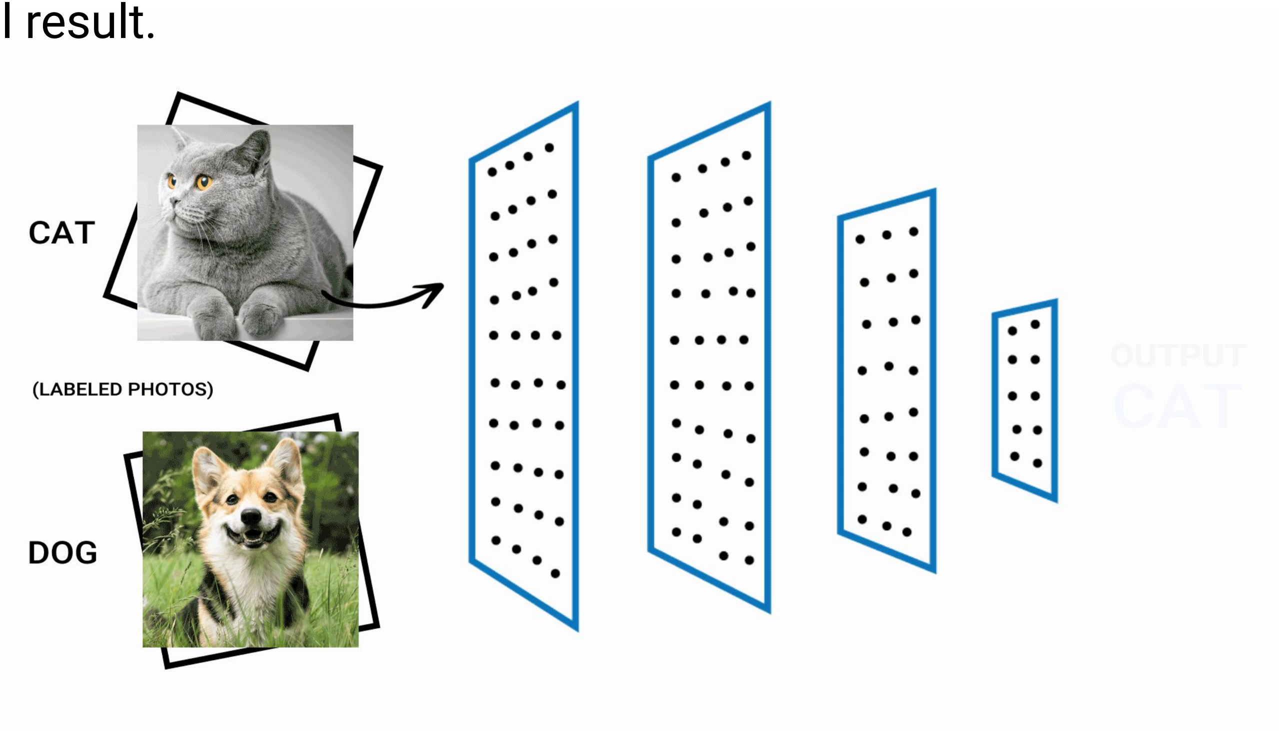
Neural networks are a set of algorithms that are modeled after the human brain. They're designed to recognize patterns and interpret sensory data through a kind of machine perception, labeling, or clustering raw inputs.

What Is a Neural Network?



What Is a Neural Network?

The layers of neurons are connected and weighed against one another until the neurons reach the final, outer layer. The final layer returns a numerical or encoded categorical result.



Neural Networks

Neural networks are particularly useful in data science because they serve multiple purposes:

01

One of the most popular uses for a neural network is as a classification algorithm to determine how to categorize an input.

02

Another popular use for a neural network is as a regression model to predict dependent outputs from independent input variables.

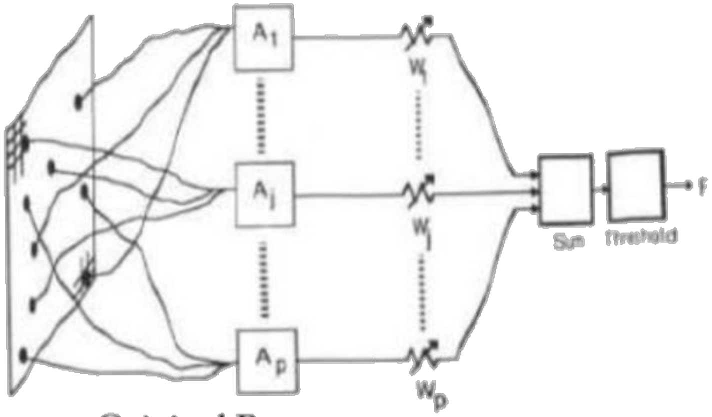
Neural Network Basics

Perceptron, the Computational Neuron

A **Perceptron** is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn, and it processes elements in the training set one at a time.

Perceptron

The perceptron was introduced by Frank Rosenblatt in 1957. He proposed a perceptron learning rule based on the original MCP neuron conceived by McCulloch and Pitts in 1943.

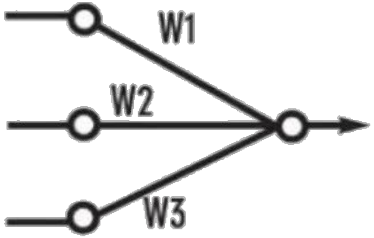


Original Perceptron

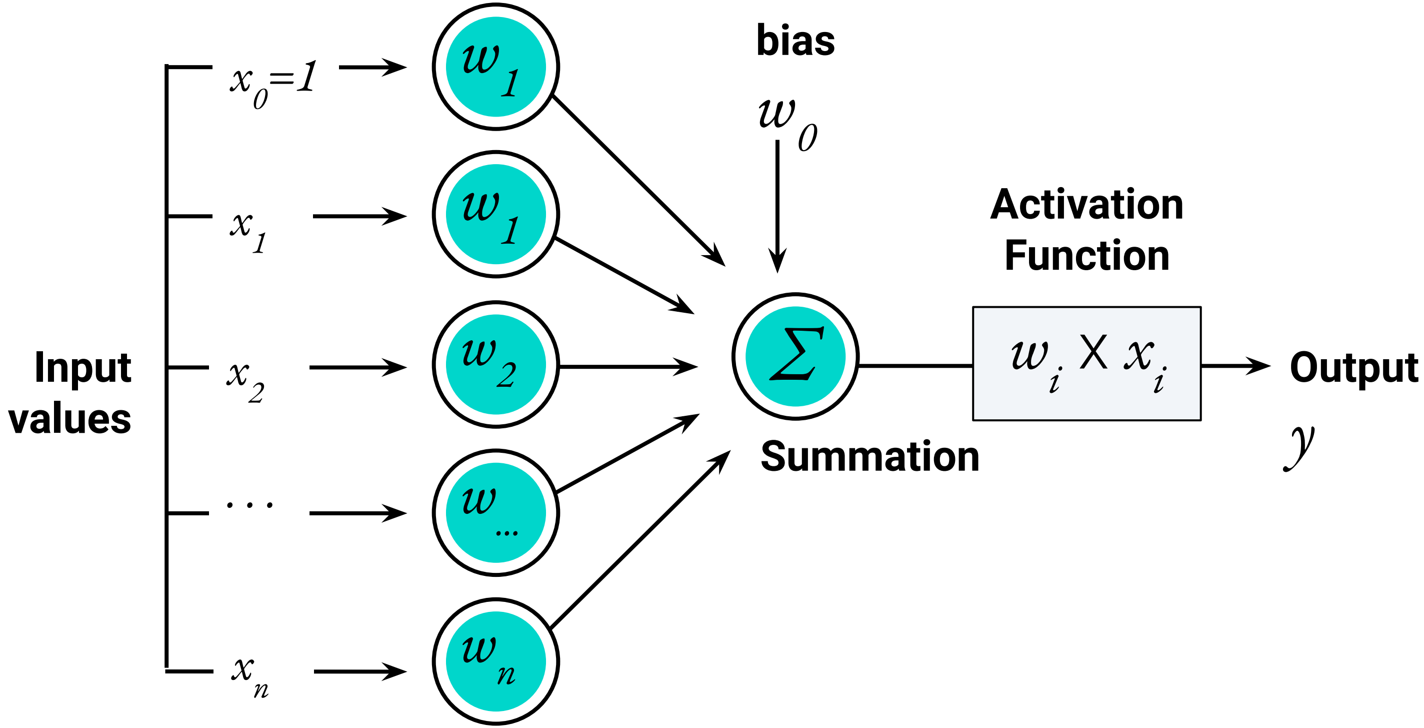
(From Perceptrons by M. L. Minsky and S. Papert, 1969, Cambridge, MA: MIT Press. Copyright 1969 by MIT Press.

Frank Rosenblatt
(1928-1971)

Simplified model:

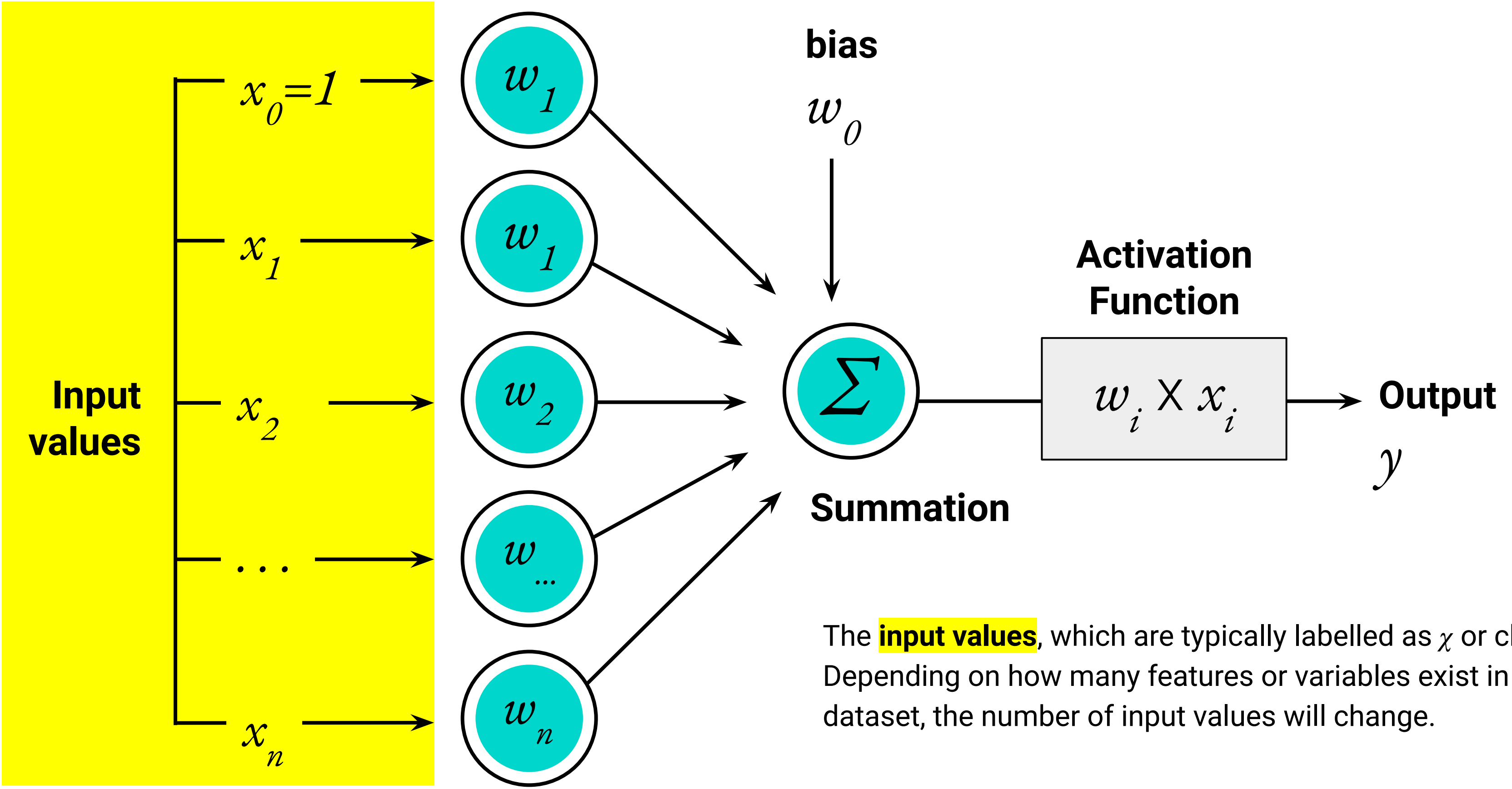


Perceptron Model



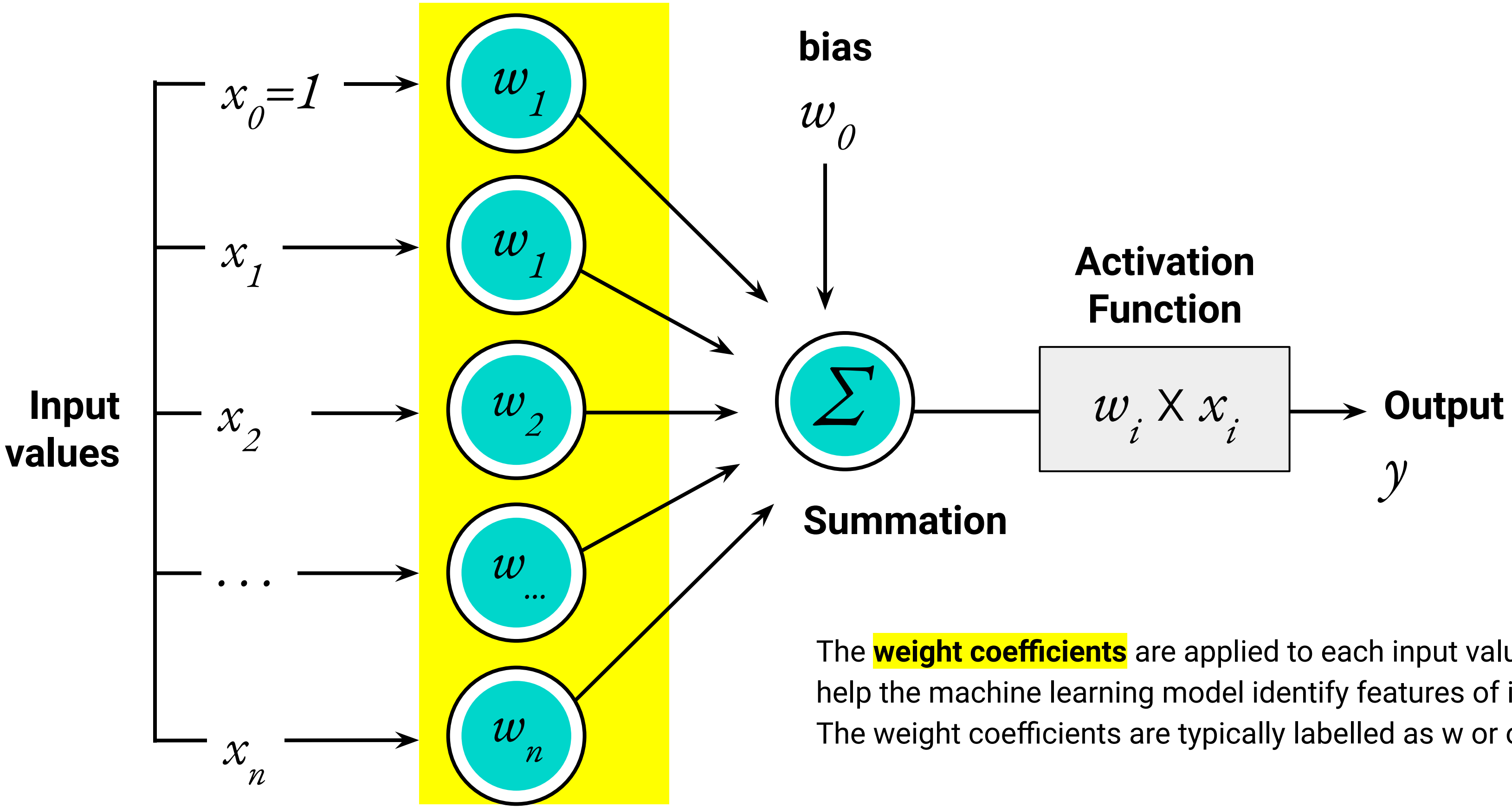
The perceptron model has four major components

Input Values (shown as *x*'s)



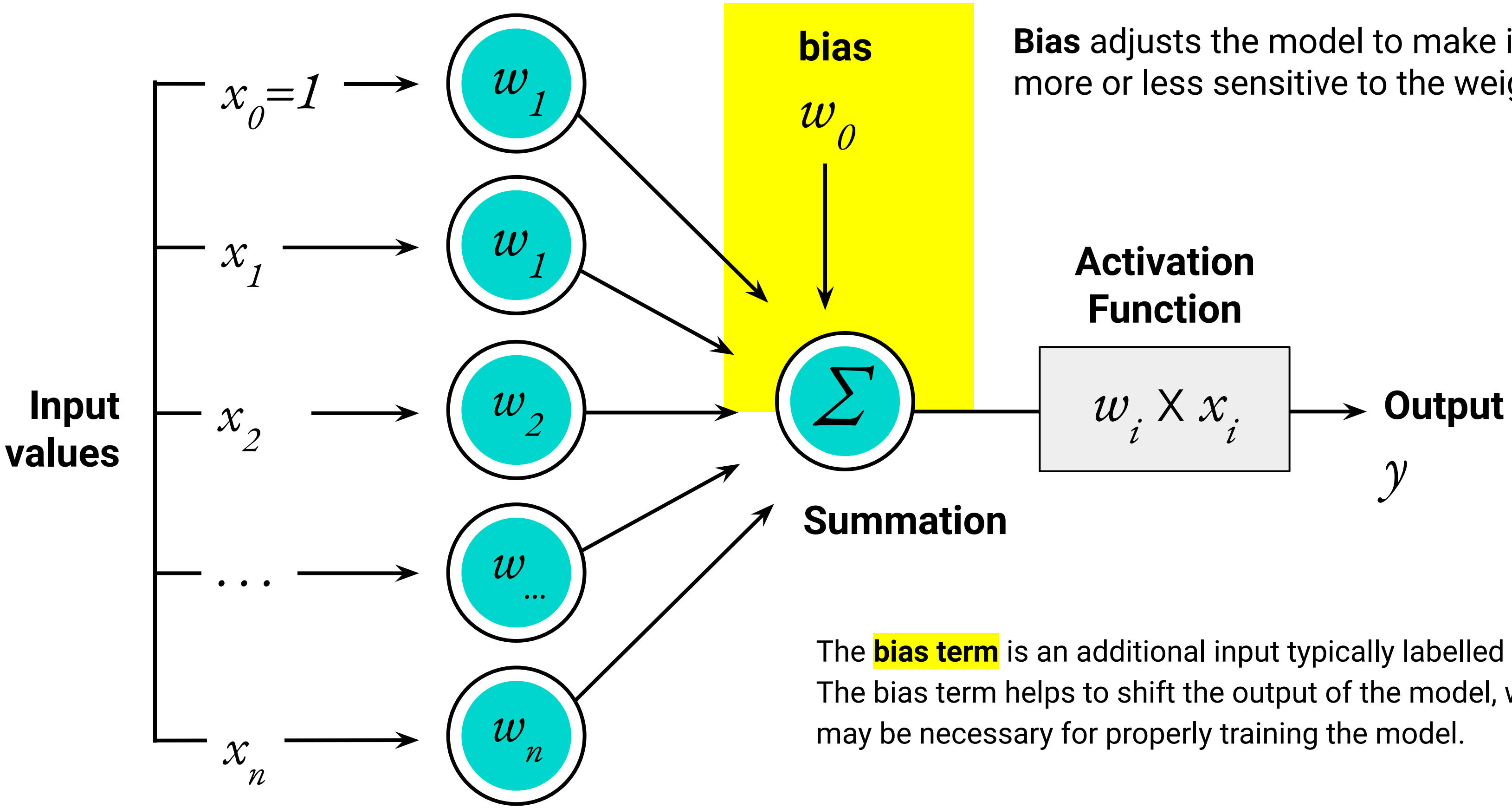
The **input values**, which are typically labelled as χ or chi. Depending on how many features or variables exist in the dataset, the number of input values will change.

Weight Coefficients (denoted as *w*'s)



The **weight coefficients** are applied to each input value to help the machine learning model identify features of interest. The weight coefficients are typically labelled as w or omega.

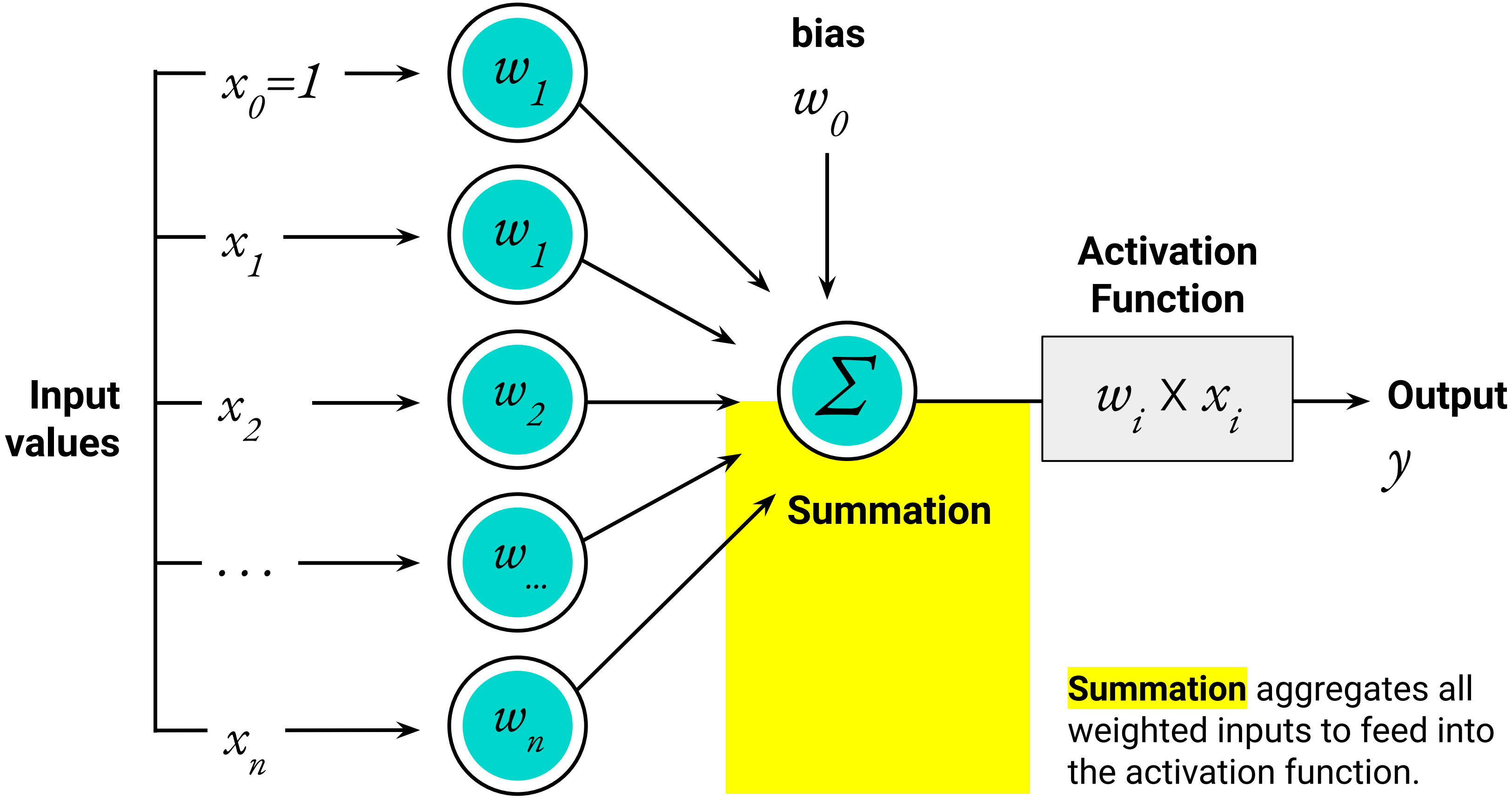
A Constant Value Called Bias



Bias adjusts the model to make it either more or less sensitive to the weights.

The **bias term** is an additional input typically labelled as ω. The bias term helps to shift the output of the model, which may be necessary for properly training the model.

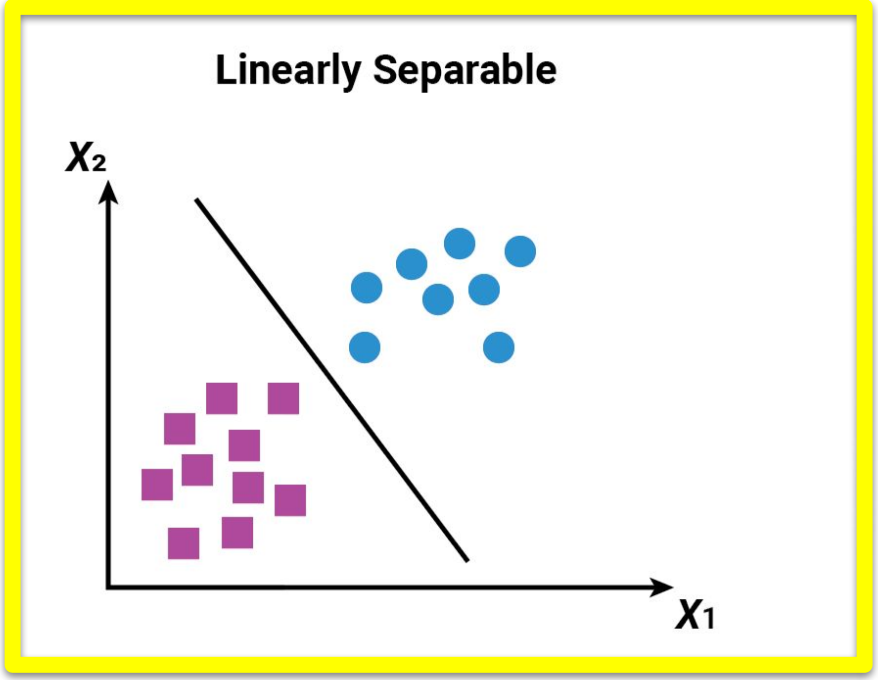
Net Summary Function (the Summation)

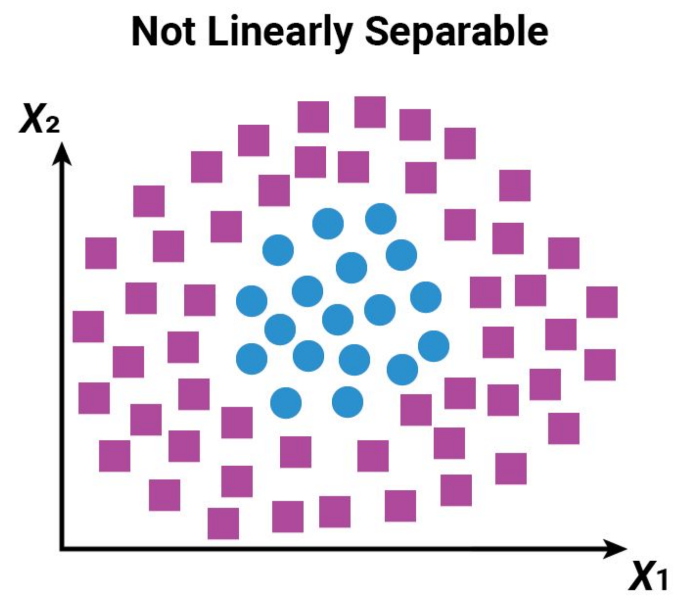


Summation aggregates all weighted inputs to feed into the activation function.

Perceptron, the Computational Neuron

The perceptron, also known as the **linear binary classifier**, is most commonly used to separate data into two groups.

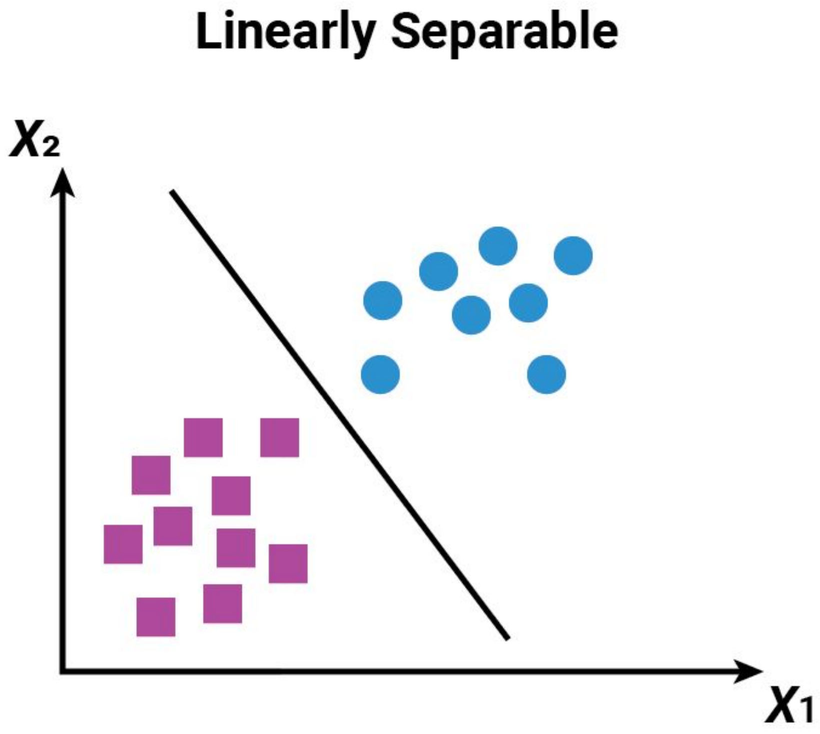




Perceptron, the Computational Neuron

Consider linearly separable data: the perceptron algorithm separates and classifies the data into two groups using a linear equation

Since we provide the model with our input features and parameters, the perceptron model is a form of **supervised machine learning**.

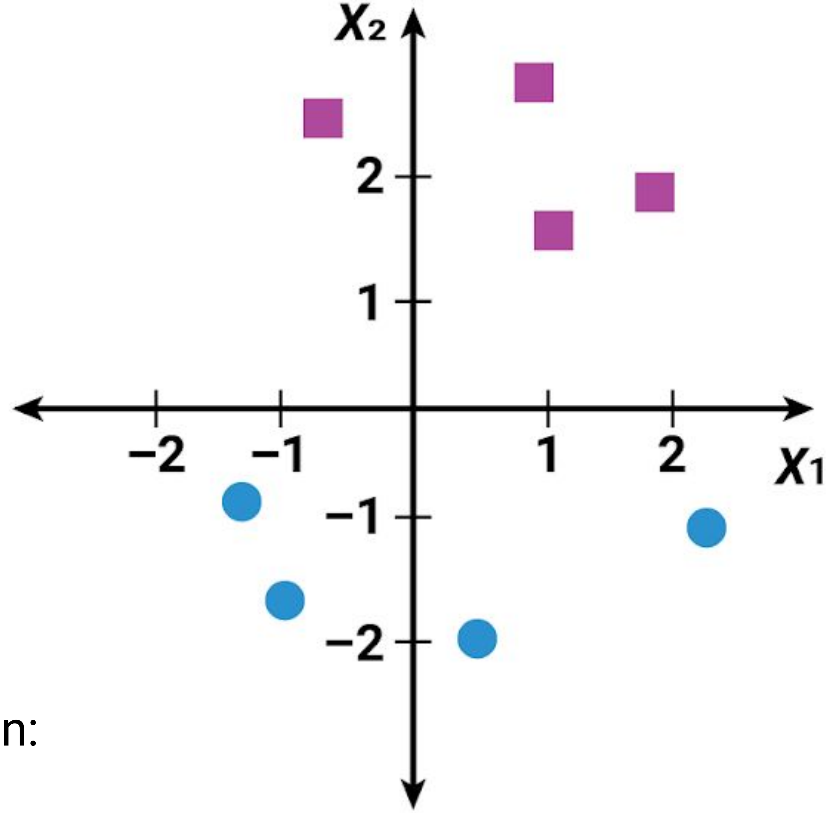


Perceptron Example

Our model will try to classify values in a two-dimensional space; our perceptron model will use three inputs:

$\chi 1$	the *x* value
$\chi 2$	the *y* value
$\omega 0$	the bias constant

The end result of our two-dimensional perceptron model is the net sum function: $\omega 0 + \chi 1 \omega 1 + \chi 2 \omega 2$.

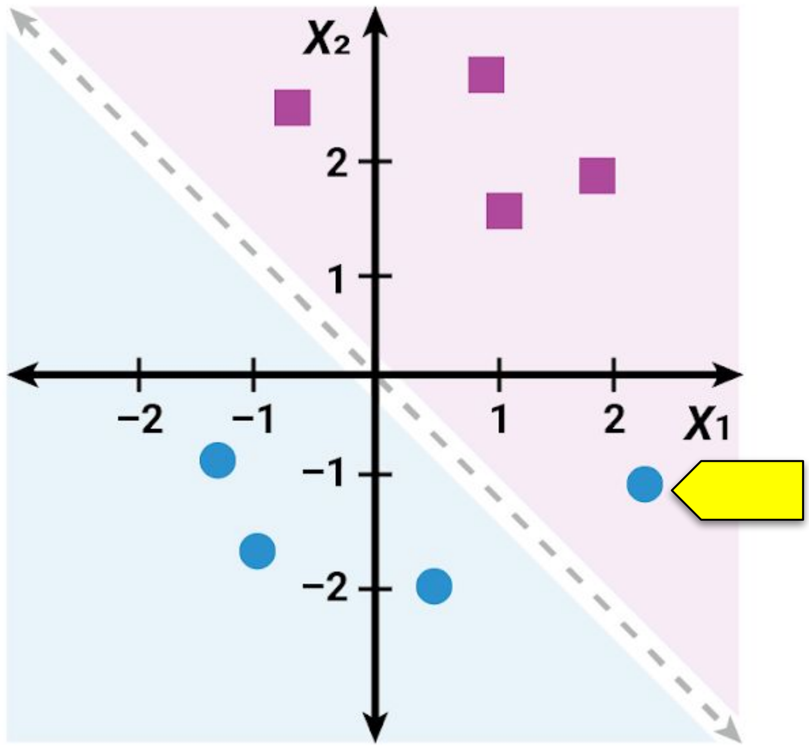


As with any untrained machine learning model, the weights and coefficients are arbitrary and oftentimes random.

Untrained Perceptron Model on Our Dataset

An untrained model almost classified the two groups perfectly.

It misclassified one blue circle.

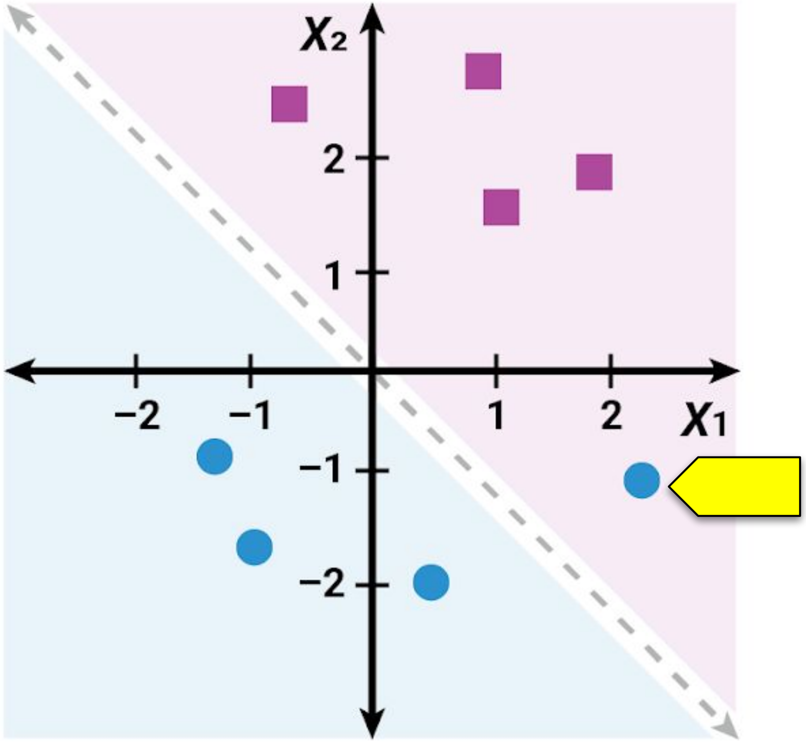


Untrained Perceptron Model on our Dataset

The perceptron model will evaluate each data point and determine if the input weights should change.

If a data point is classified correctly, the weights will not change.

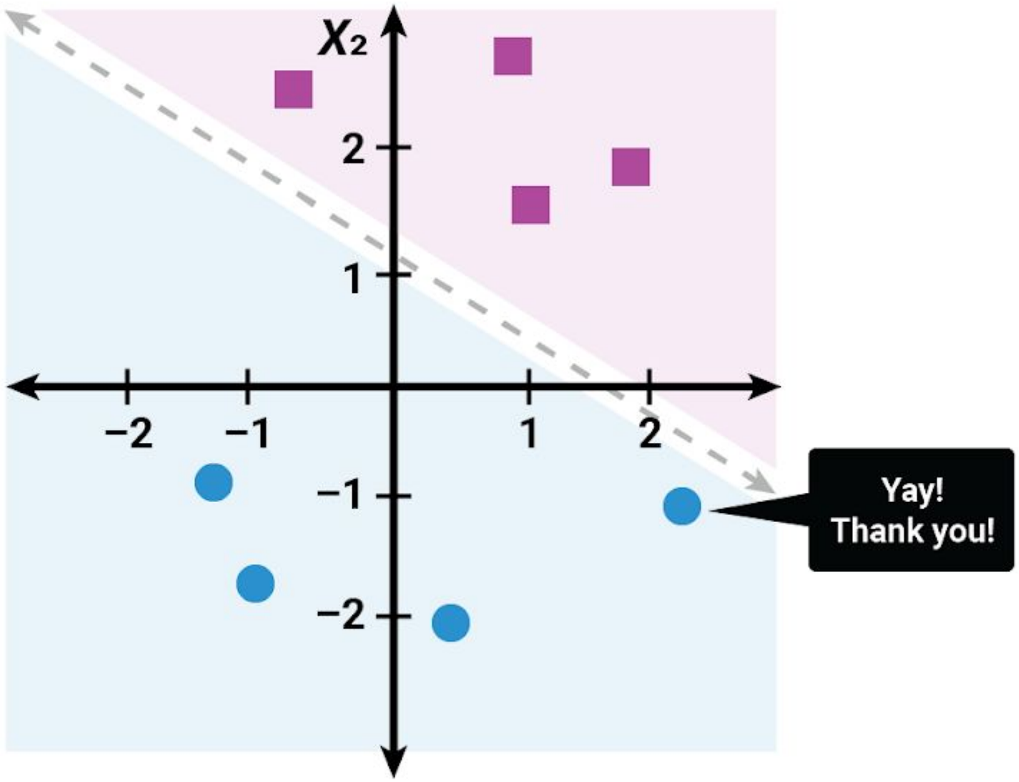
If a data point is misclassified, the weights will move the model closer to the missed data point.



Trained Perceptron Model on our Dataset

Each data point is evaluated to determine if the input weights should change.

Since a data point is misclassified, the weights will move the model closer to the missed data point.



Perceptron Model Training

As with other machine learning algorithms and models, perceptron model training continues until one of three conditions is met:

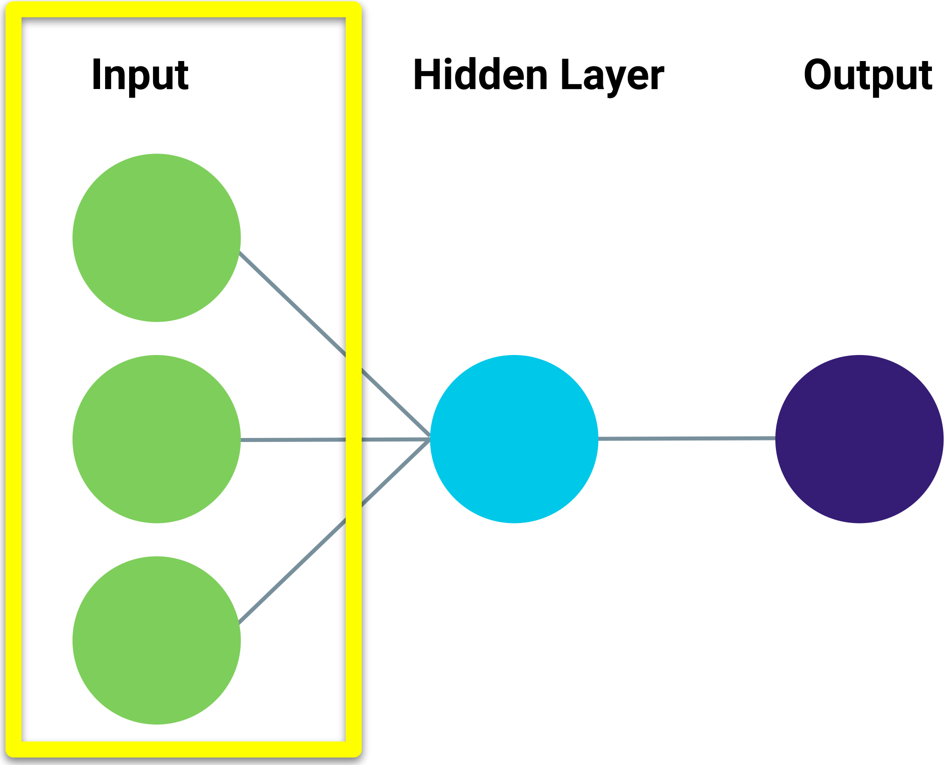
The perceptron model exceeds a predetermined performance threshold set by the designer before training. In machine learning, this is quantified by minimizing the loss metric.	For example, if we are working with noisy data that cannot be preprocessed or excluded, our model may not be able to exceed a certain level of performance without overfitting. Therefore, we would want to set a training cutoff at the point of model convergence.
The perceptron model training performs a set number of iterations determined by the designer before training.	For example, if we know roughly how many iterations it takes for a model to achieve desired performance, we can just "set it and forget it" to train over a specific interval.
The perceptron model is stopped or encounters an error during training.	This will typically be a hardware or power issue as long as our input data goes through cleaning and preprocessing before use. If we set up our model to save itself after a specific number of training iterations, we can resume training immediately.

A simple perceptron model is very similar to our basic statistical models. However, the power of the perceptron model comes from its ability to handle multidimensional data and interact with other perceptron models.

Make the Connections in a Neural Network

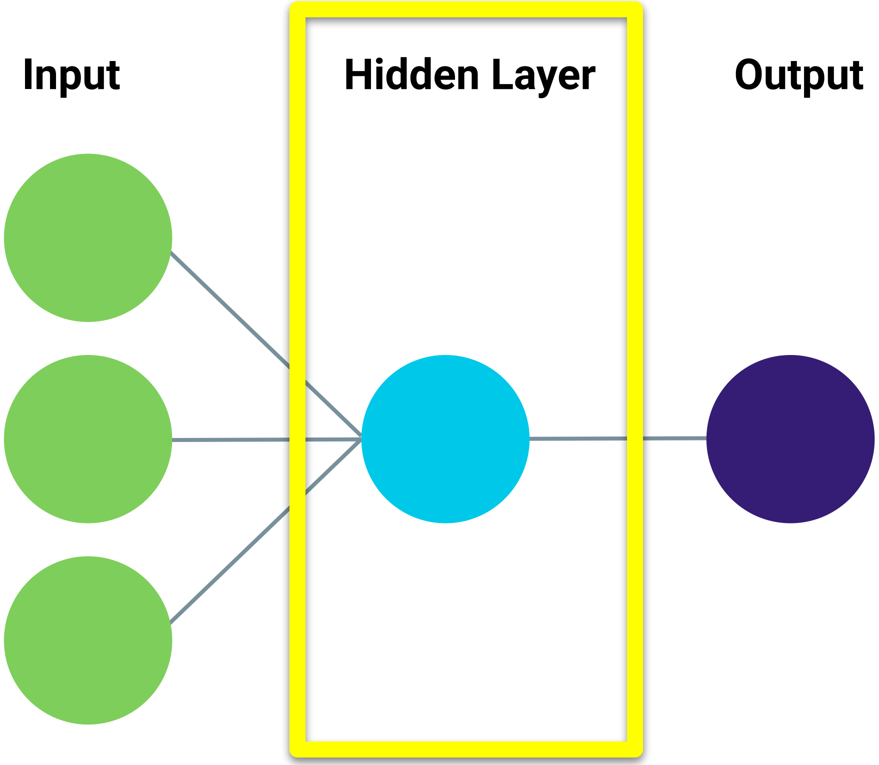
The Structure of a Neural Network

An **input layer** of input values (transformed by weight coefficients)



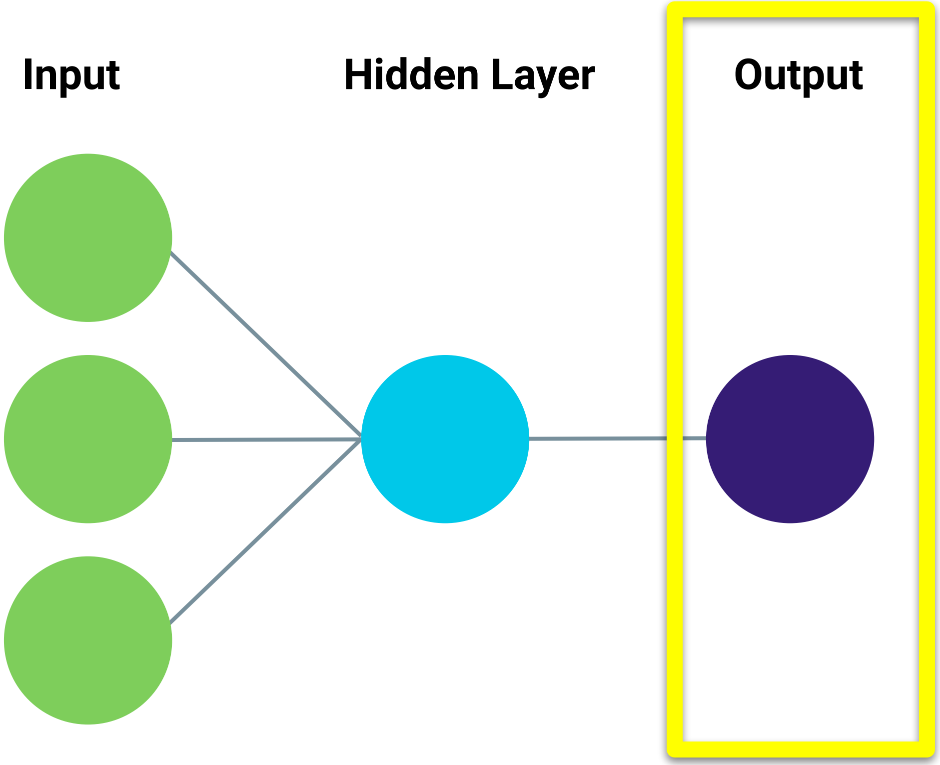
The Structure of a Neural Network

A single **hidden layer** of neurons (single neuron or multiple neurons)



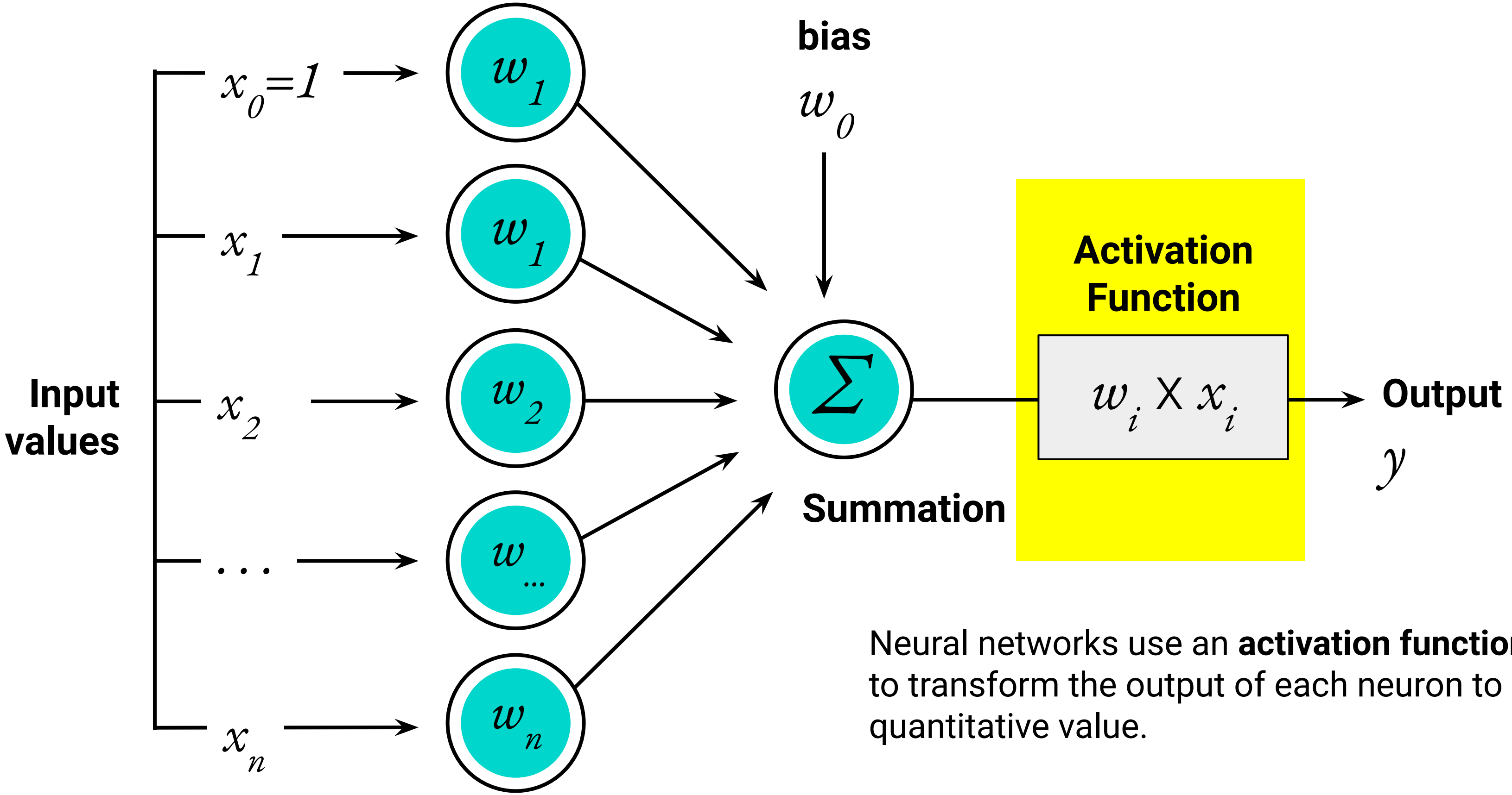
The Structure of a Neural Network

An **output layer** reports the classification or regression value



If each neuron has its own output, how does the neural network combine each neuron's output into the model's classification or regression output?

Activation Function (Transformation After Summation)

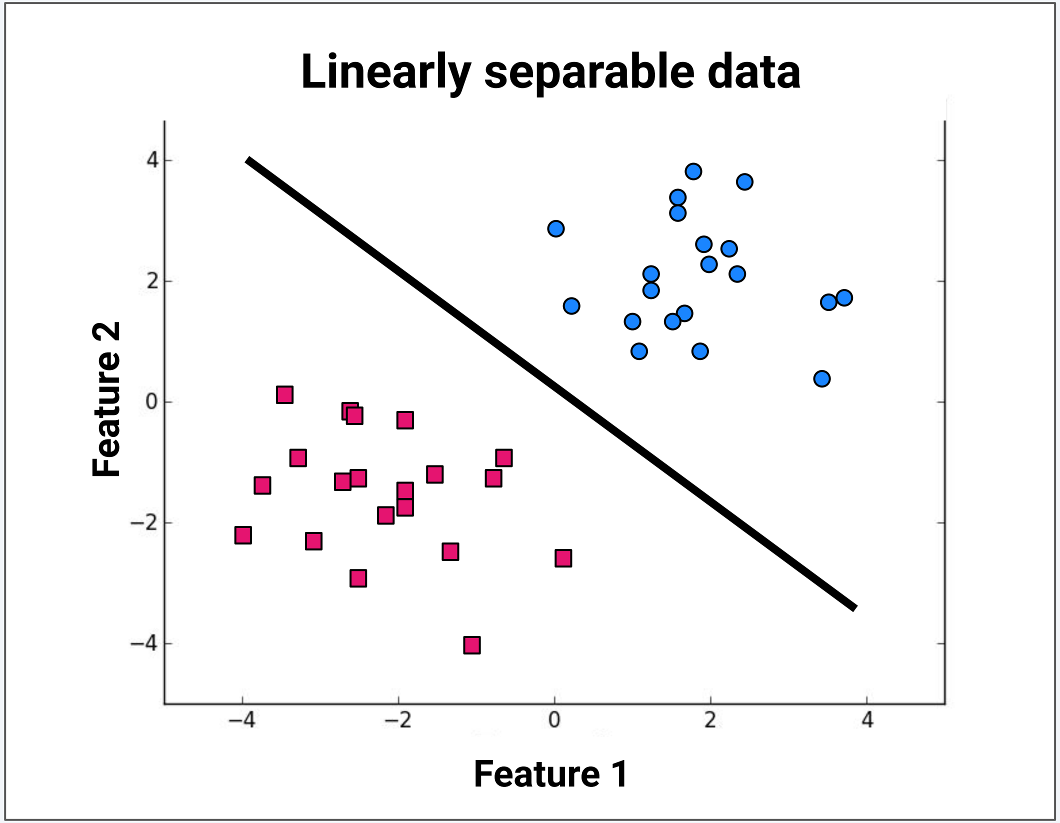


Neural networks use an **activation function** to transform the output of each neuron to a quantitative value.

There are a variety of activation functions that can be used for many specific purposes. However, most neural networks will use one of the following activation functions.

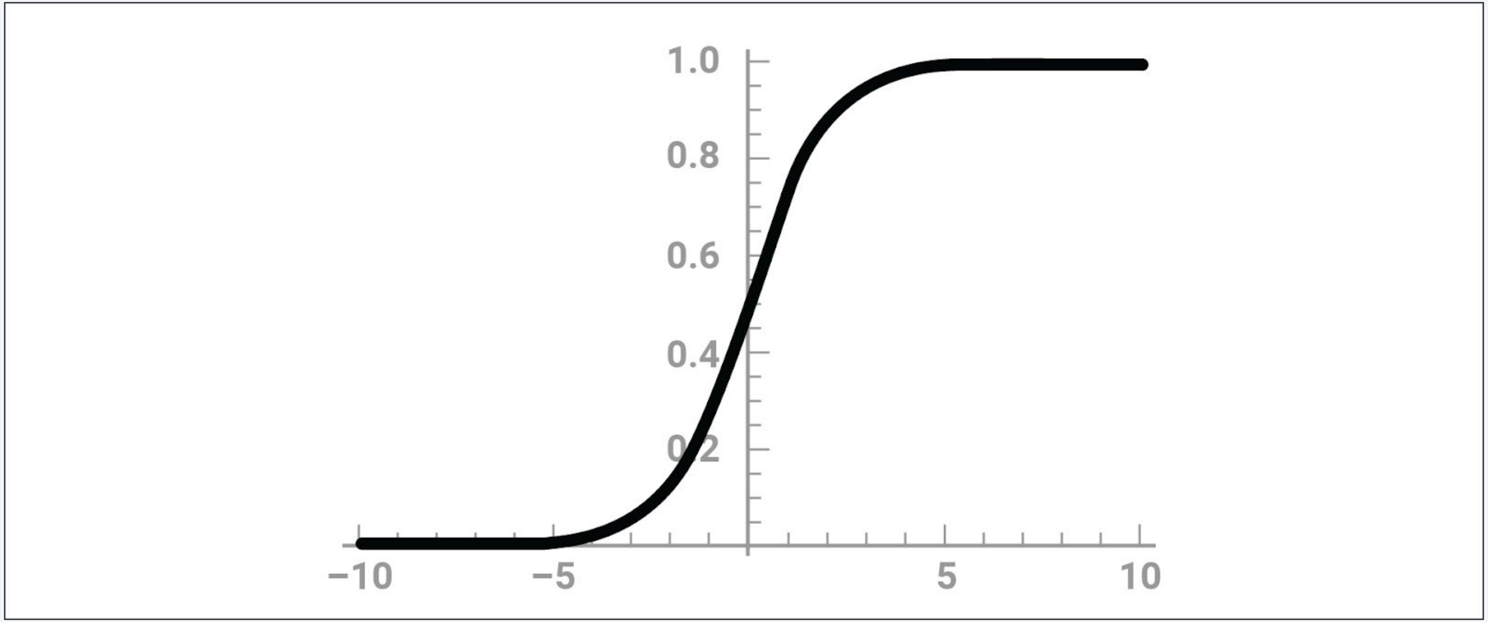
Linear Function

The **linear function** transforms the output into the coefficients of a linear model (the equation of a line).



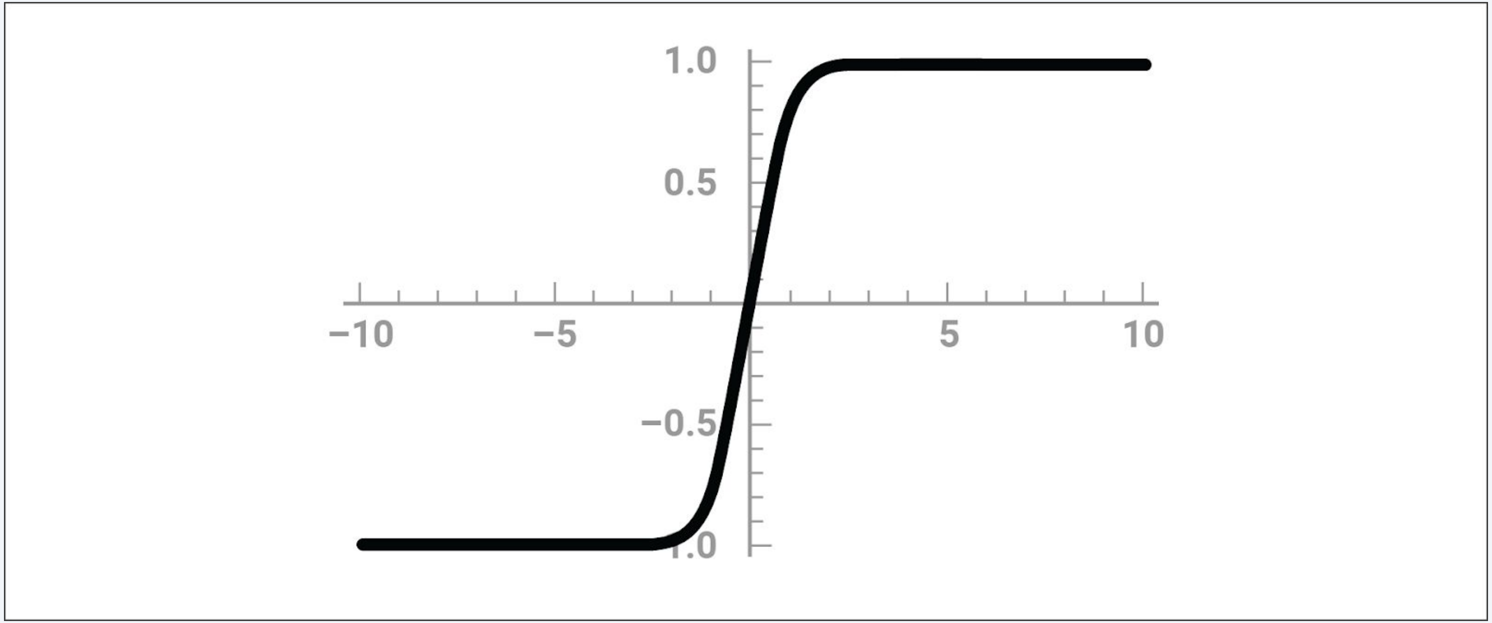
Sigmoid Function

The **sigmoid function** transforms the neuron's output to a range between 0 and 1, which is especially useful for predicting probabilities. A neural network that uses the sigmoid function will output a model with a characteristic S-curve.



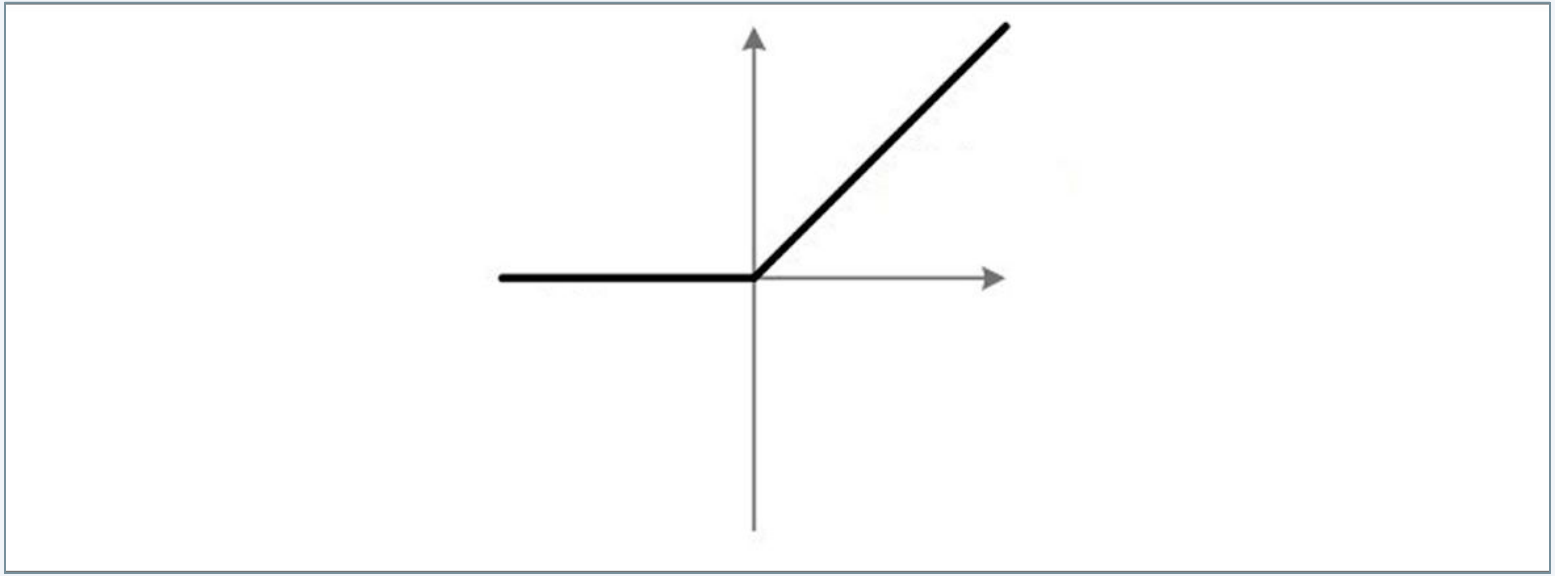
Tanh Function

The **tanh function** transforms the output to a range between –1 and 1.
The output for a model using a tanh function also forms a characteristic S-curve.
It’s primary use is classifying data into one of two classes.



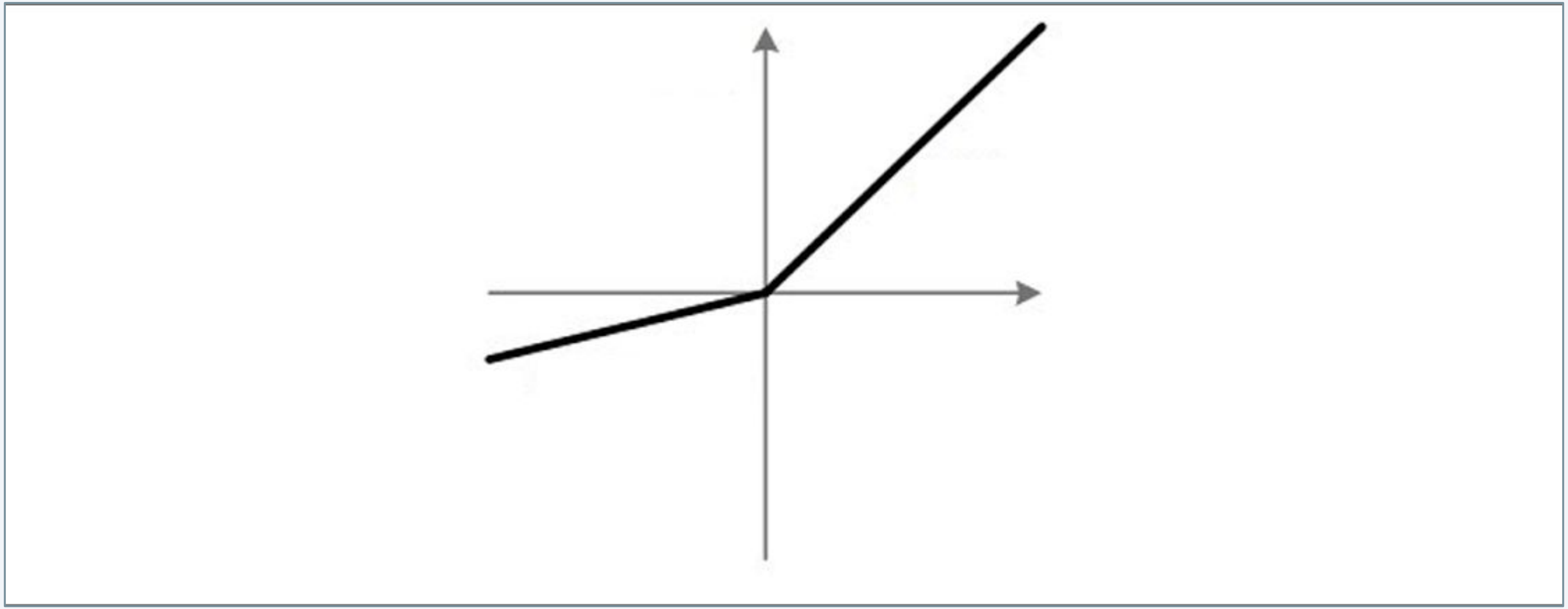
Rectified Linear Unit (ReLU)

The **rectified linear unit (ReLU)** function returns a value from 0 to infinity. This activation function transforms any negative input to 0. It is the most commonly used activation function in neural networks due to its faster learning and simplified output. However, it is not always appropriate for simpler models.



Leaky ReLU Function

The **leaky ReLU function** is an alternative to the ReLU function, which can sometimes work better. Instead of transforming negative input values to 0, it transforms them into much smaller negative values.



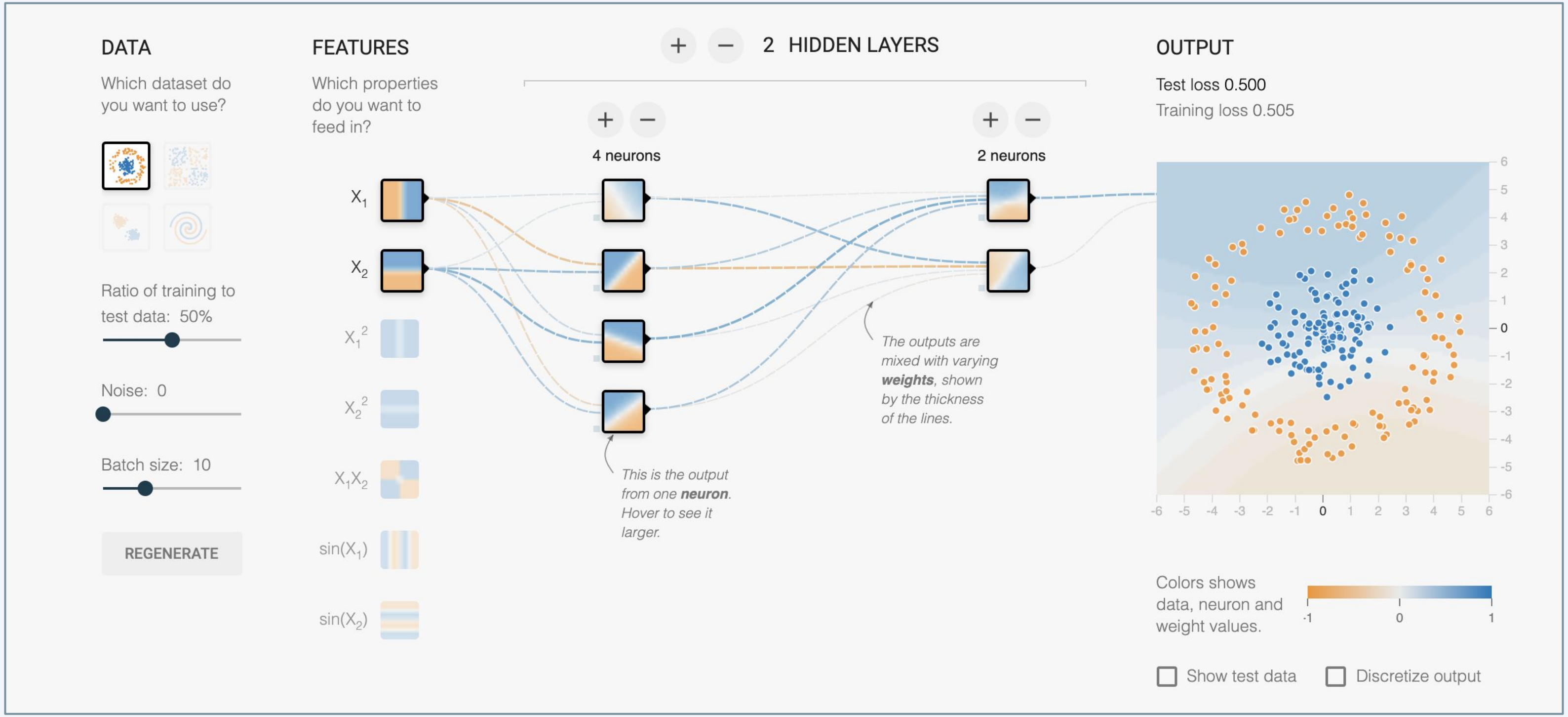
TensorFlow Playground

We are going to use the TensorFlow Playground to build our Neural Networks.

TensorFlow Playground is an application developed by TensorFlow as a teaching tool to “*demystify the black box*” of neural networks.

TensorFlow Playground

The application provides a working simulation of a neural network as it trains on a variety of different datasets and conditions.



TensorFlow Playground

We can also use TensorFlow Playground to test different configurations of our neural network models, like an abstract form of our model-fit-predict workflow.

Time to Code

Playing in TensorFlow Playground

Suggested Time:

15 minutes

TensorFlow can be picky about what version of Python it works with, as well as what modules need to be installed, so we will use Google Colab to run our notebooks in the cloud.

Google Colab

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with:

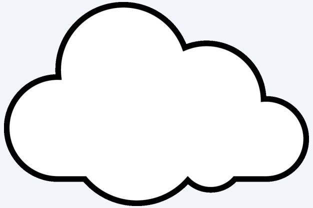
- Zero configuration required
- Free access to GPUs
- Easy sharing

Google Colab

Advantages of Google Colab

01

Google Colab is a cloud-based notebook, different from Jupyter Notebook, which runs locally on our machine.



02

Cloud-based notebooks are user friendly and won’t require any type of module installation.

03

TensorFlow was also developed by Google, which makes running in Colab fairly seamless.

Instructor Demonstration

Set up Google Colab

Understanding the TensorFlow Neural Network Structure

There are a number of smaller modules within the TensorFlow 2.0 library that make it even easier to build machine learning models.

For our purposes, we'll use the Keras module to help build our neural networks

Keras contains multiple classes and objects that can be combined to design a variety of neural network types.

Keras

For a basic neural network, we will use two Keras classes:

Sequential class

The Sequential class is a linear stack of neural network layers where data flows from one layer to the next. This class is what we simulated in the TensorFlow Playground.

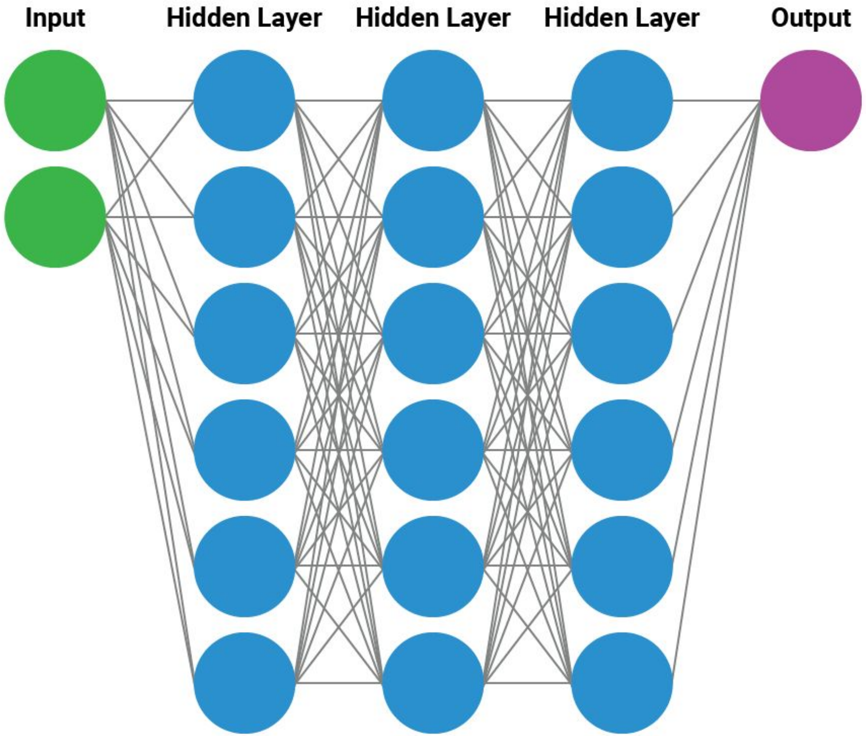
Dense class

The generalized Dense class allows us to add layers within the neural network.

Keras

For the **Sequential model**, we will add multiple Dense layers that will act as our input, hidden, and output layers.

For each **Dense layer**, we'll define the number of neurons and activation functions.



Keras

Finally, once we have completed the Sequential model design, we will apply the same scikit-learn model -> fit -> predict/transform workflow we have used previously.

Hello, Neural Network World

Time to Code

Work through a Neural Network Workflow

Suggested Time:

15 minutes

Activity: BYONNM

(Build Your Own Neural Network Model)

In this activity, you will implement your own basic classification neural network model using the TensorFlow Keras module. In addition, you will create your own dummy data, split the data into training and test sets, and normalize the data using scikit-learn.

Suggested Time:

15 minutes

Activity: BYONNM (Build Your Own Neural Network Model)

Instructions

- Using the starter code provided, visualize the blobs dummy dataset using a Pandas scatter plot.
- Randomly split the dummy data into training and test datasets using scikit-learn's `train_test_split` method.
- Normalize both datasets using scikit-learn's StandardScaler class.
- Create a basic neural network with 5 neurons in the hidden layer using the Keras module.
  Note: Your neural network should use two inputs and produce one classification output.
- Compile your basic neural network model.
- Train the neural network model over 50 epochs.
- Evaluate the performance of your model, printing your test loss metric and the predictive accuracy of the model on the test dataset.

Bonus

- Try creating a new neural network with a different number of neurons.
- Train the new neural network model on the same training data, and test the performance on the same testing dataset.
- Create a line plot that visualizes the neural network predictive accuracy over each epoch.

Time's Up! Let's Review.

Debrief on Neural Networks

Debrief on Neural Networks

Regardless of what machine learning model or technology we use, we follow the same general workflow:

01 Decide on a model and create a model instance.

02 Split into training and testing sets.

03 Preprocess the data

04 Train/fit the training data to the model.

05 Evaluate the model for predictions and transformations.

Debrief on Neural Networks

When creating a neural network, we use two Keras classes:

Sequential class

`Sequential` is a linear stack of neural network layers where data flows from one layer to the next.

Dense class

`Dense` allows us to add layers within the neural network.

In the dense class, we add *at least one* input layer where we define the activation function, and we add an output layer that uses a probability activation function.

Dense Class

Then, we:

- Create a summary to get the structure of the `Sequential` model.
- Compile the `Sequential` model.
- Fit the model to the training data with a defined number of epochs.
- Evaluate the model using the test data.

Questions?