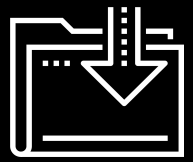
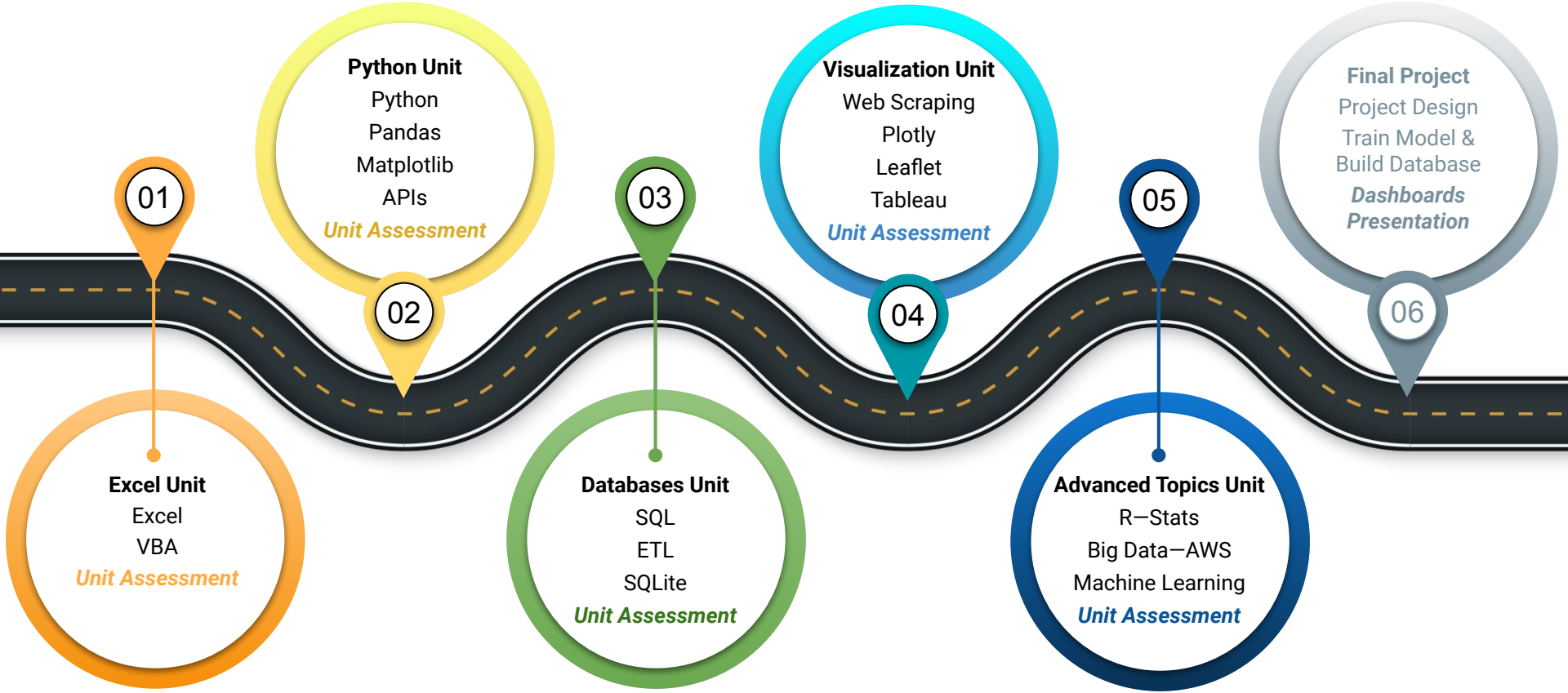
Applying Advanced Machine Learning to Real Data

Data Boot Camp

Lesson 19.2



The Big Picture



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.

Today's Agenda

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

01 Implementing deep neural network models using TensorFlow

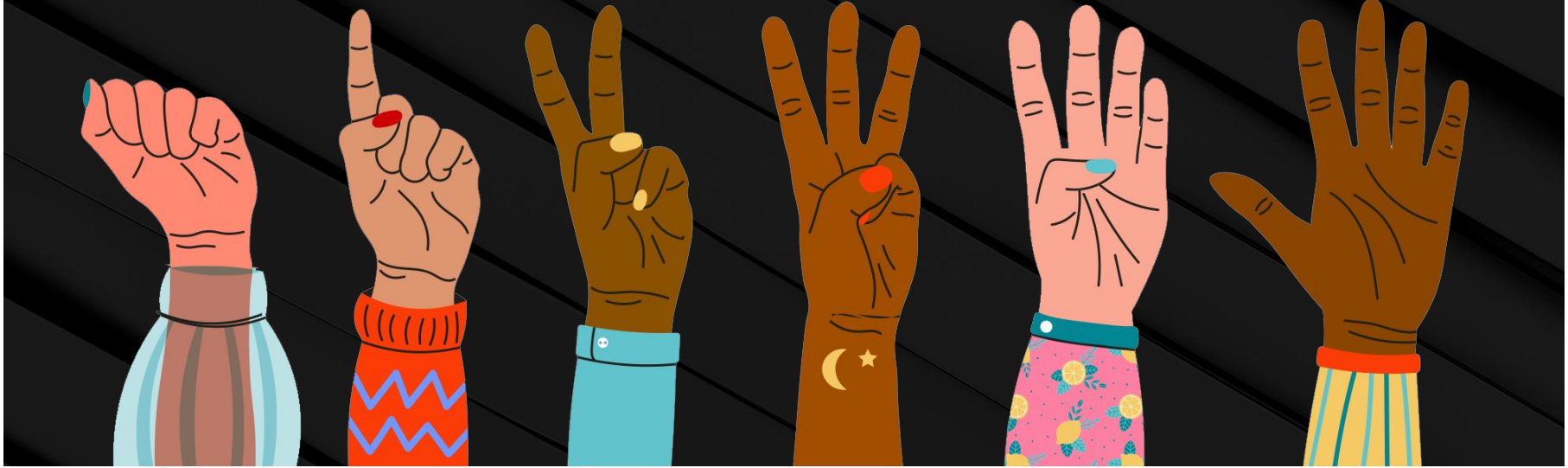
02 Increasing the performance of a neural network

03 Applying deep learning to datasets

Make sure you've downloaded any relevant class files!

FIST TO FIVE:

How comfortable do you feel with this topic?



We Must Dig Deeper... into Neural Networks

Time to Code

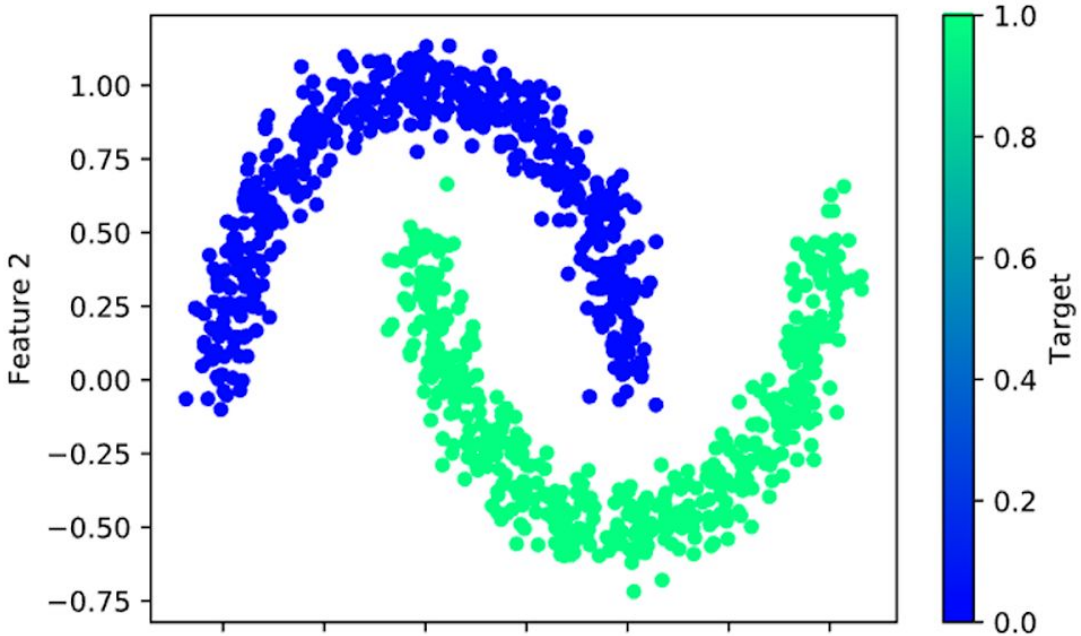
Over the Moon on Basic Neural Networks

Suggested Time:

5 minutes

Over the Moon on Basic Neural Networks

Similar to the swirl input data we saw in TensorFlow Playground, the moons dataset, made from scikit-learn's `make_moons` method, is not linearly separable.



What are the next steps in the workflow?

Over the Moon on Basic Neural Networks

Next steps:

01 Split the dataset, then scale and standardize each feature

02 Fit the model to our training data

03 Transform the data

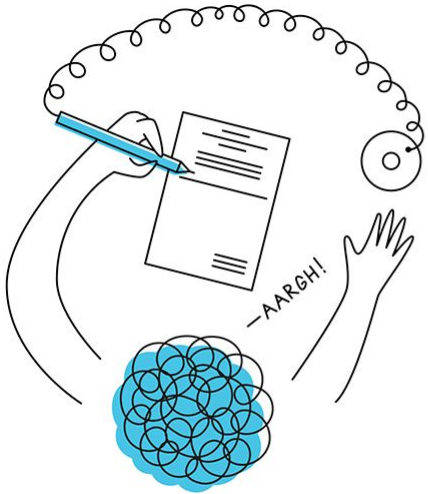
04 Create a Sequential model

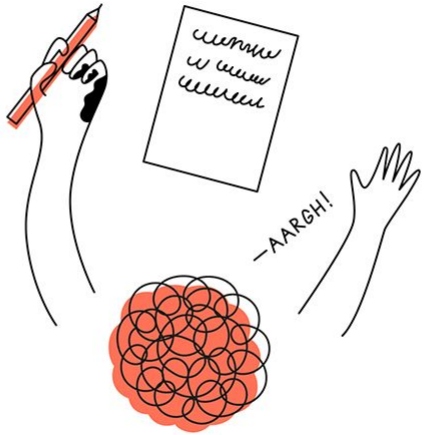
05 Add the first layer and output layer, and then get the structure of the Sequential model

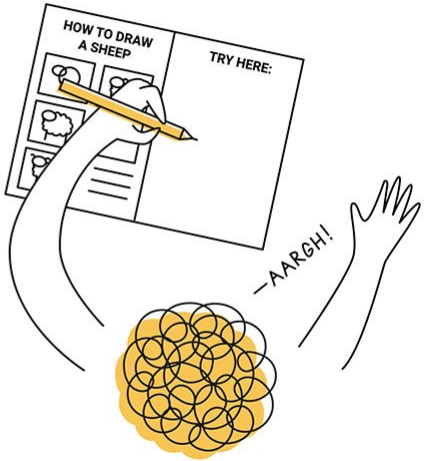
Depending on the dataset or the use case for the model, a predictive accuracy of ~85% may be sufficient for a first-pass model.

Over the Moon on Basic Neural Networks

For example, let's say we were trying to build a neural network that can predict if students are left-handed or right-handed. A model that was able to make correct predictions 85% of the time would be pretty accurate!



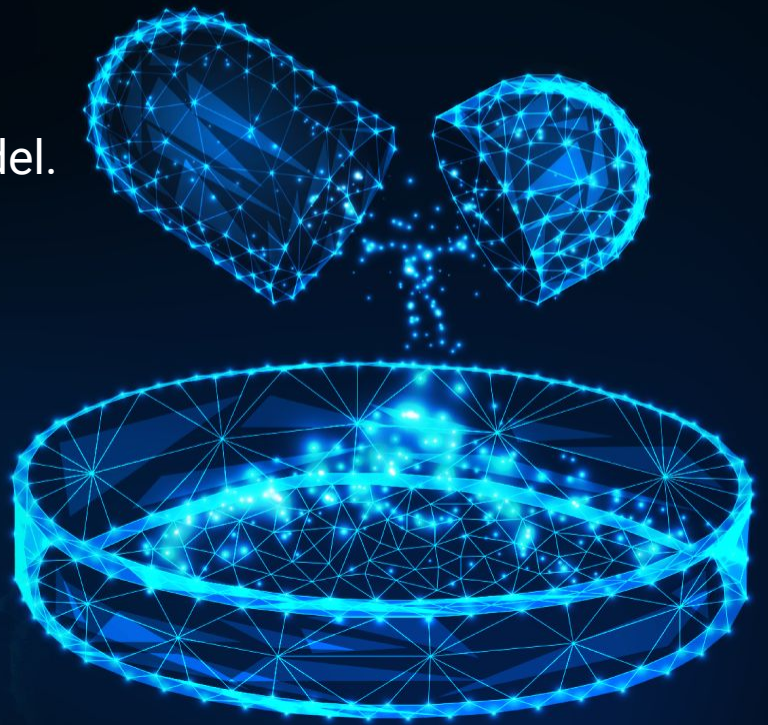




Over the Moon on Basic Neural Networks

In many industrial and medical use cases,
a machine learning model must exceed 95%,
or even 99%, classification accuracy.

In these cases, we would not accept
the basic single-neuron, single-layer model.

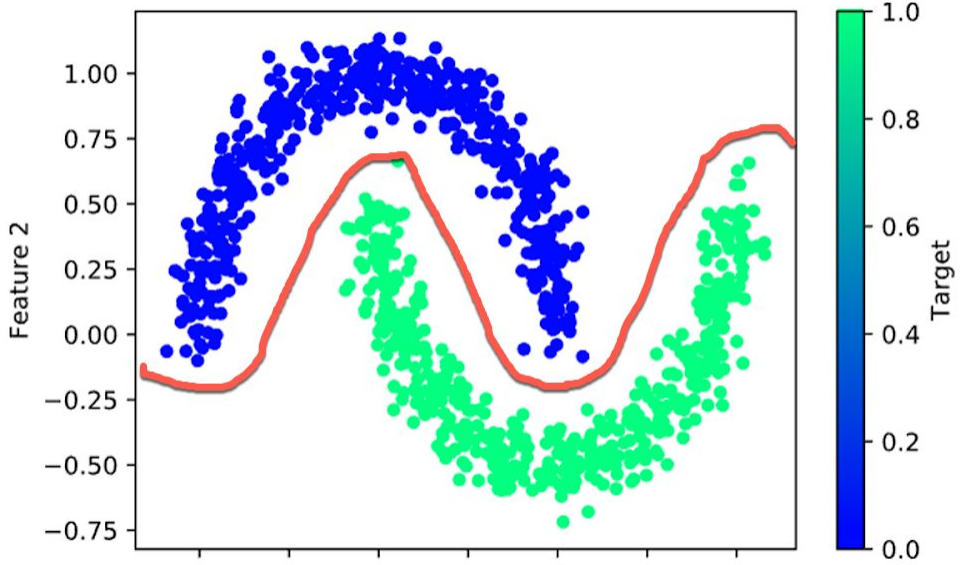


One possible solution to our performance problem is to add more neurons.

Over the Moon on Basic Neural Networks

In our dataset, our two groups are separated by drawing a complex polynomial line. Therefore, a single-layer, multiple-neuron model would still struggle to adequately classify our two groups with only two inputs.

In these cases, we must create a neural network model capable of identifying complex nonlinear relationships.

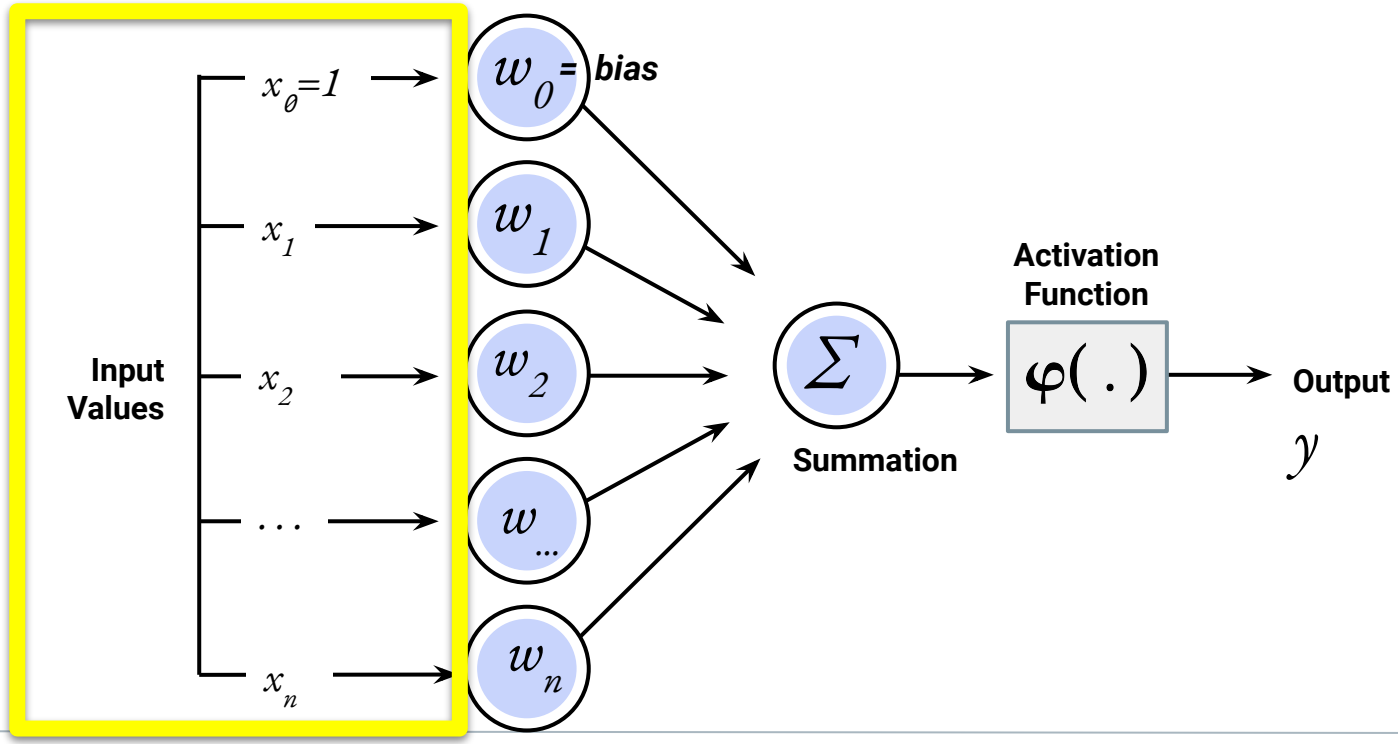


Questions?

Getting Deep with Deep Learning Models

Getting Deep with Deep Learning Models

Neural network models are designed so that input values are evaluated only one time before they are used in an output classification or regression equation.

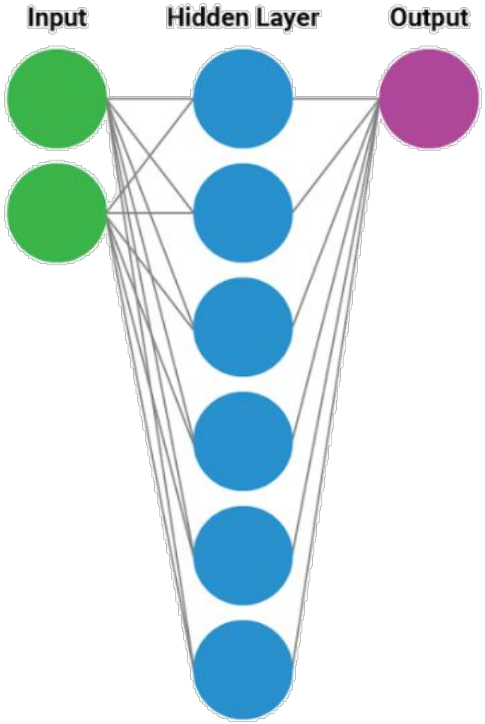


Basic neural networks are limited to interpreting simple linear relationships and data with just a few **confounding factors**, or factors that have hidden effects on more than one variable.

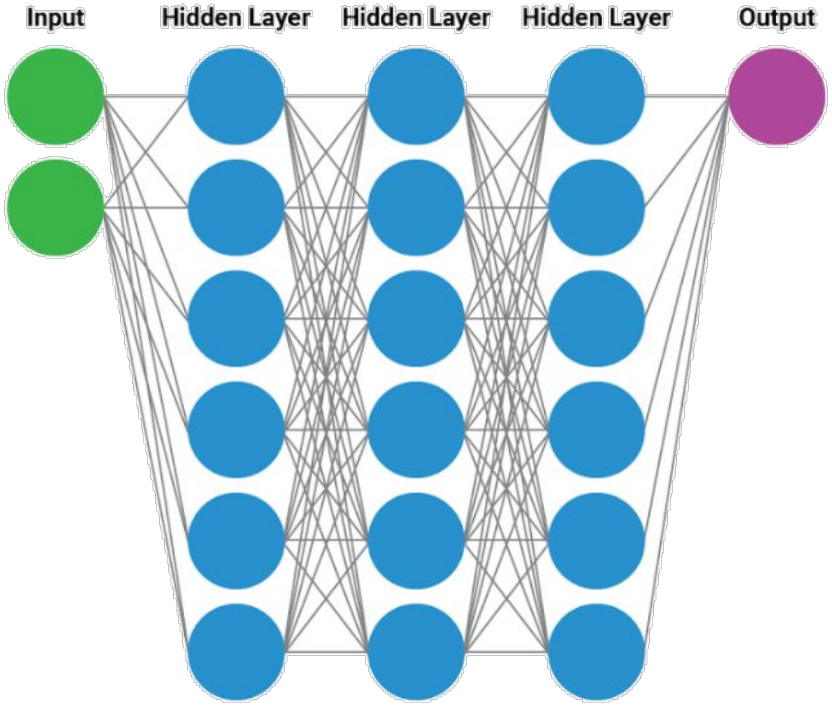
In order to address and overcome the limitations of the basic neural network, we can implement a more robust neural network model by adding additional hidden layers.

Getting Deep with Deep Learning Models

Basic Neural Network

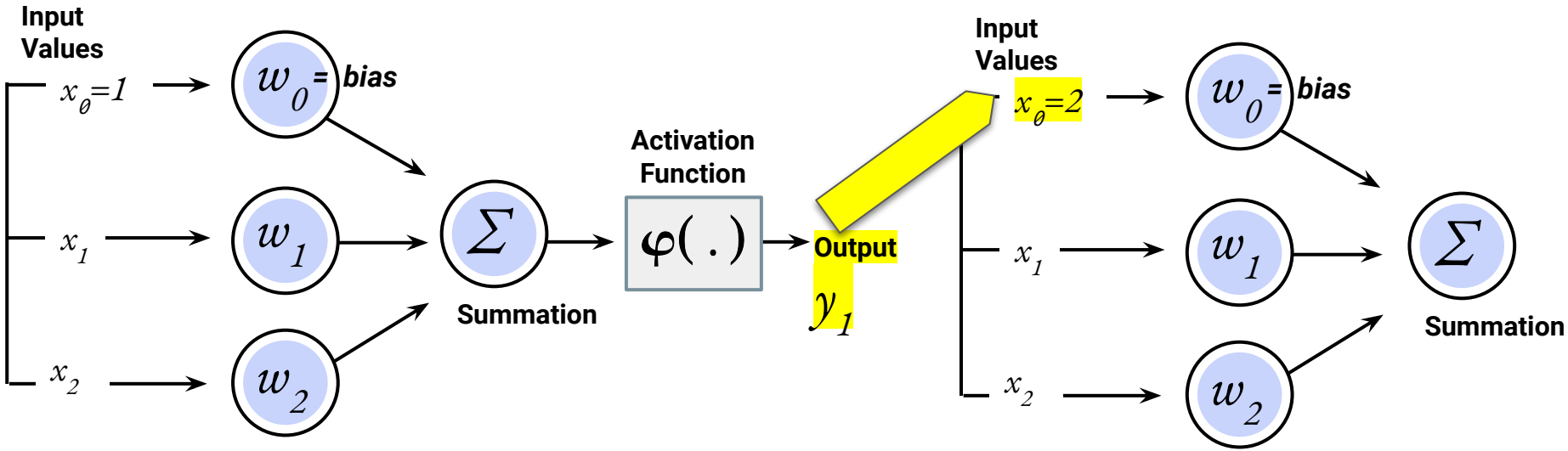


Deep Learning Model



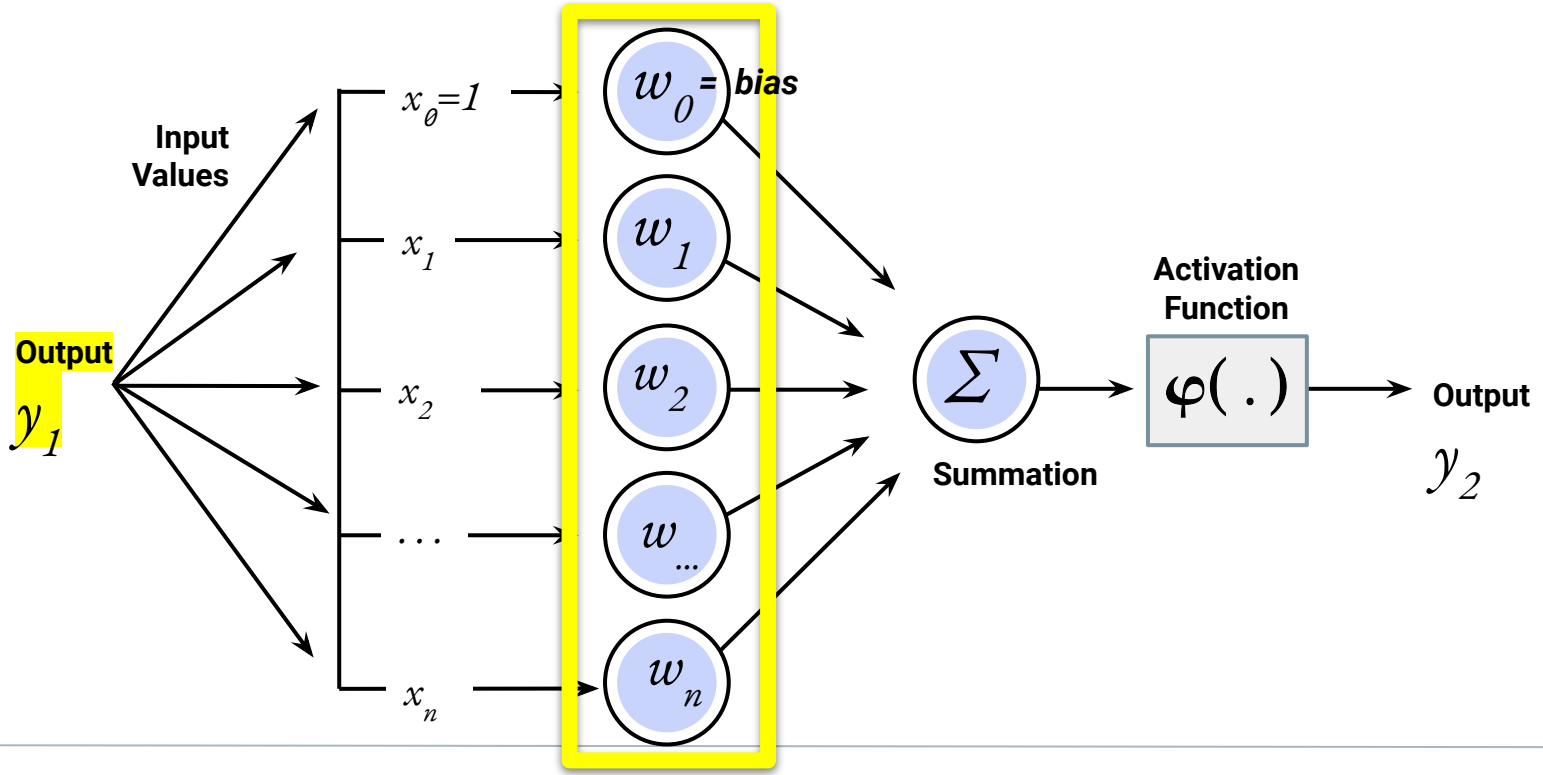
Getting Deep with Deep Learning Models

Deep learning models function similarly to the basic neural network with one major exception: the outputs of one hidden layer become the inputs to additional hidden layers of neurons.



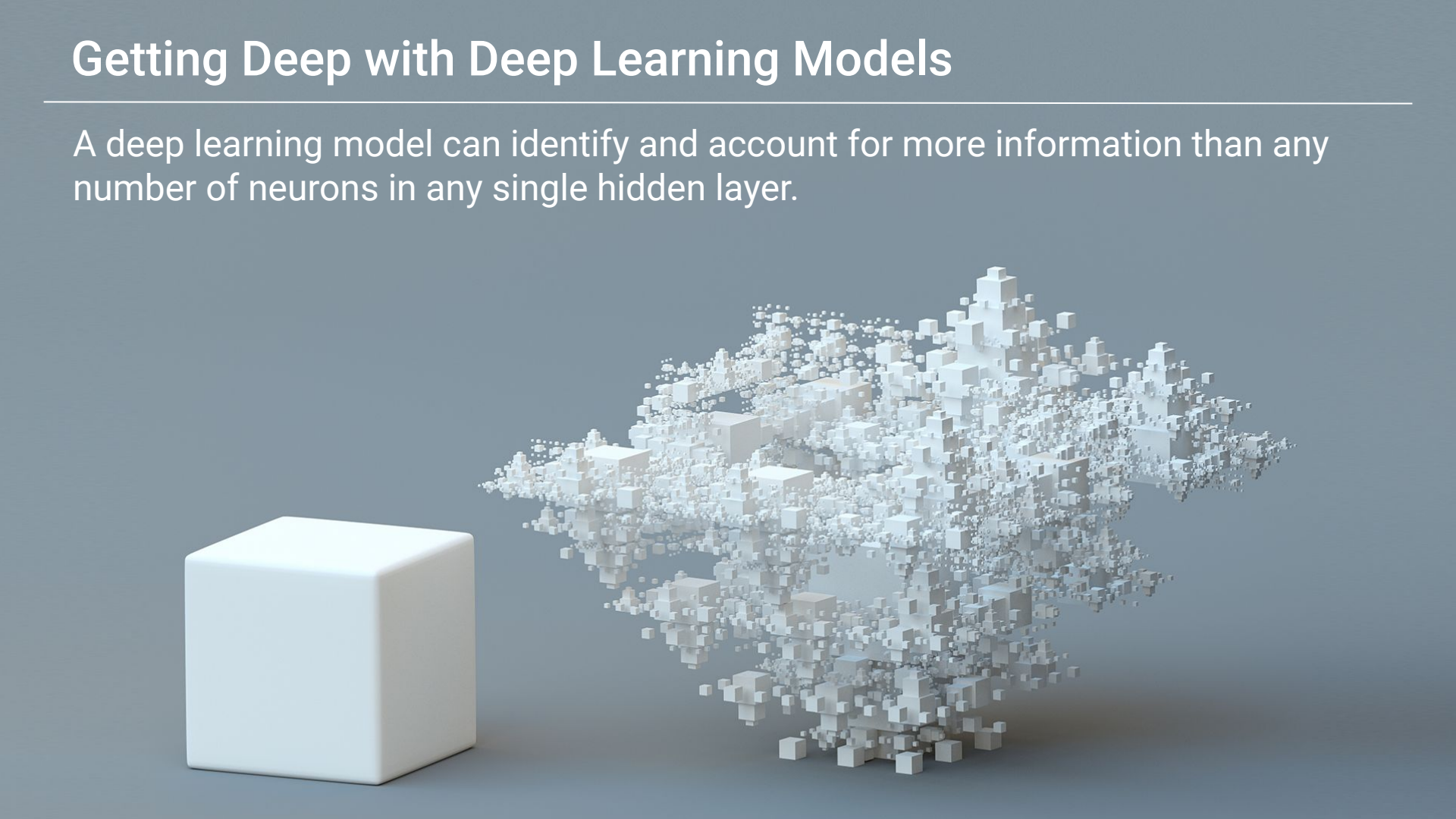
Getting Deep with Deep Learning Models

As a result, the next layer of neurons can evaluate higher-order interactions between weighted variables to identify complex, nonlinear relationships.



Getting Deep with Deep Learning Models

A deep learning model can identify and account for more information than any number of neurons in any single hidden layer.



Deep learning models got their name from their ability to learn from example data, regardless of the complexity or type of data.

Getting Deep with Deep Learning Models

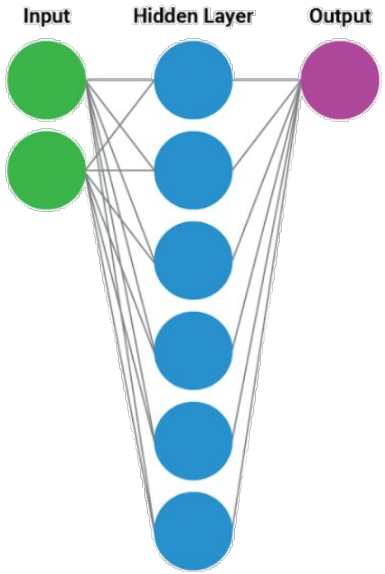
Although the numbers are constantly debated, many data scientists believe that even the most complex interactions can be characterized by as few as three hidden layers.

Deep learning models can train on images, natural language data, soundwaves, and even traditional tabular data, all with minimal preprocessing and direction.

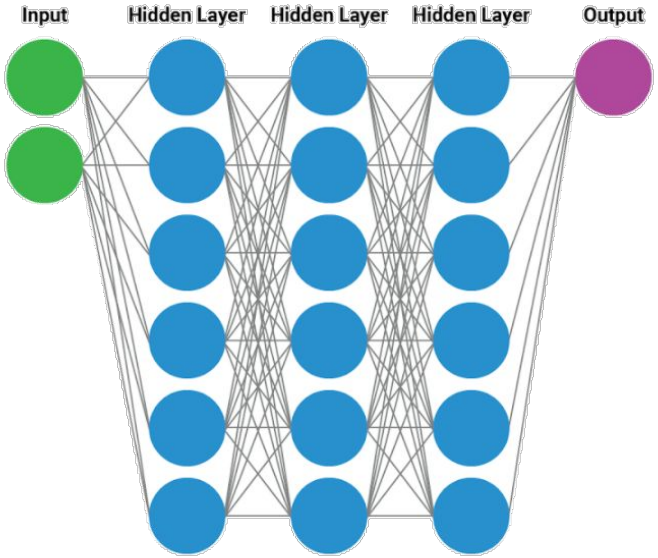
Getting Deep with Deep Learning Models

Deep learning models typically require longer training iterations and more memory resources than their basic neural network counterparts in order to achieve higher degrees of accuracy and precision.

Basic Neural Network

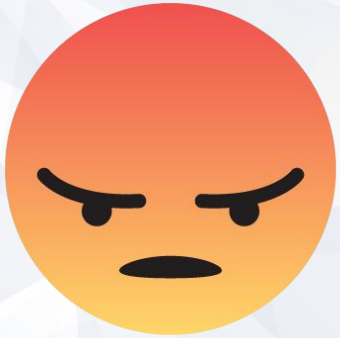


Deep Learning Model



Getting Deep with Deep Learning Models

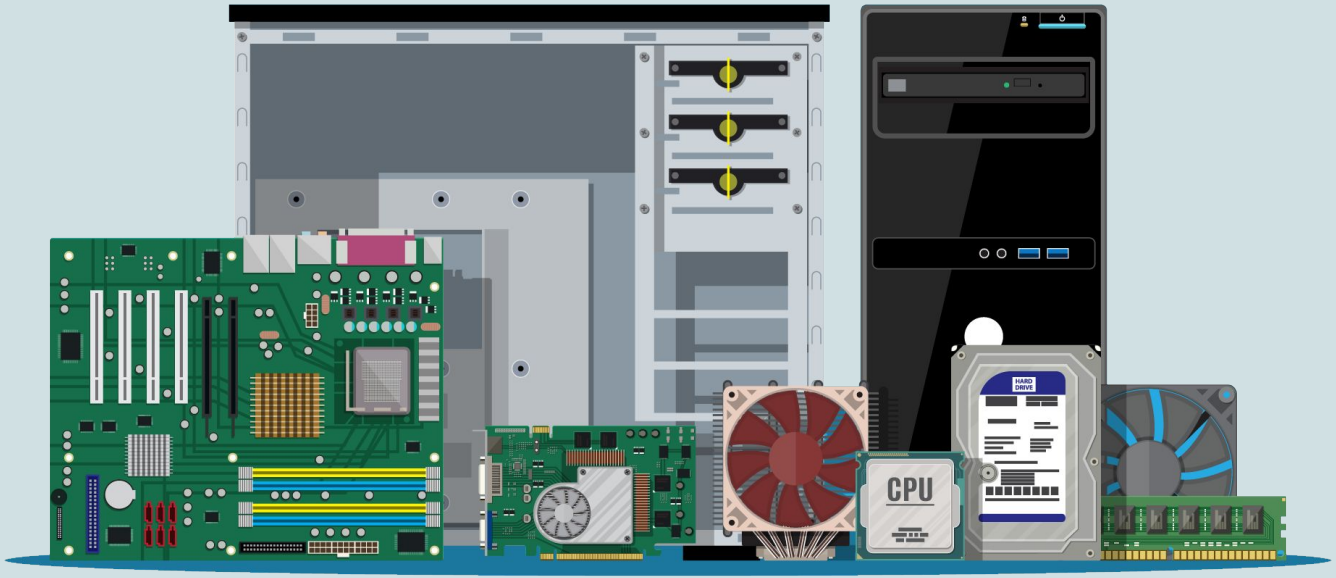
Deep learning models may have more upfront costs, but they also have higher performance potential.



Drawbacks to building a deep learning model with too many layers

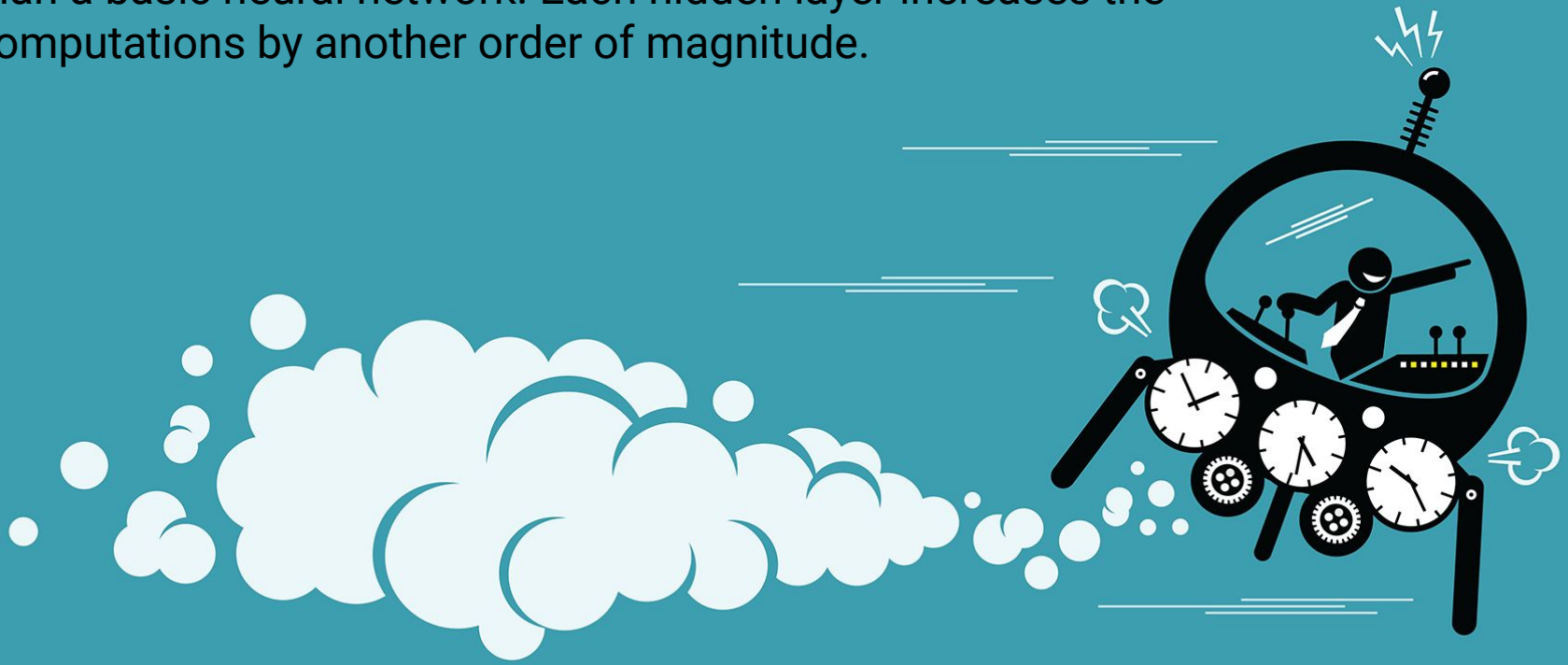
Drawbacks

Deep learning models require more and more computational resources—such as memory and CPU power—for each layer. If we have limited resources, like time, a larger deep learning model may be infeasible.



Drawbacks

A deep learning model takes considerably more time to train than a basic neural network. Each hidden layer increases the computations by another order of magnitude.



Activity: Back to the Moon

In this activity, you will try to build a deep learning classification model that can adequately predict the class from our moons dummy dataset.

Suggested Time:

15 minutes

Activity: Back to the Moon

Instructions

- Upload the starter notebook to Google Colab, then run the cells to recreate the moons dummy dataset.
- Create your Keras Sequential model and add more than one Dense hidden layer to create a deep learning model.
  Notes:
  Only your first Dense layer uses the `input_dim` parameter.
  All of your hidden layers should use the "relu" activation function.
- Compile your model and train the deep learning model on at least 100 epochs.
- Evaluate the performance of your model by calculating the loss and predictive accuracy of the model on your test dataset.

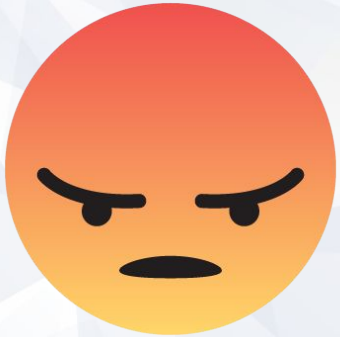
Bonus

If time permits, try recreating your deep learning model with a different set of hidden layers and neurons, and then evaluate the performance of your new model. Are you able to achieve 100% predictive accuracy?

Time’s Up! Let’s Review.

Getting the Most out of a Neural Network Model

As with any machine learning model, neural networks and deep learning models are not perfect.

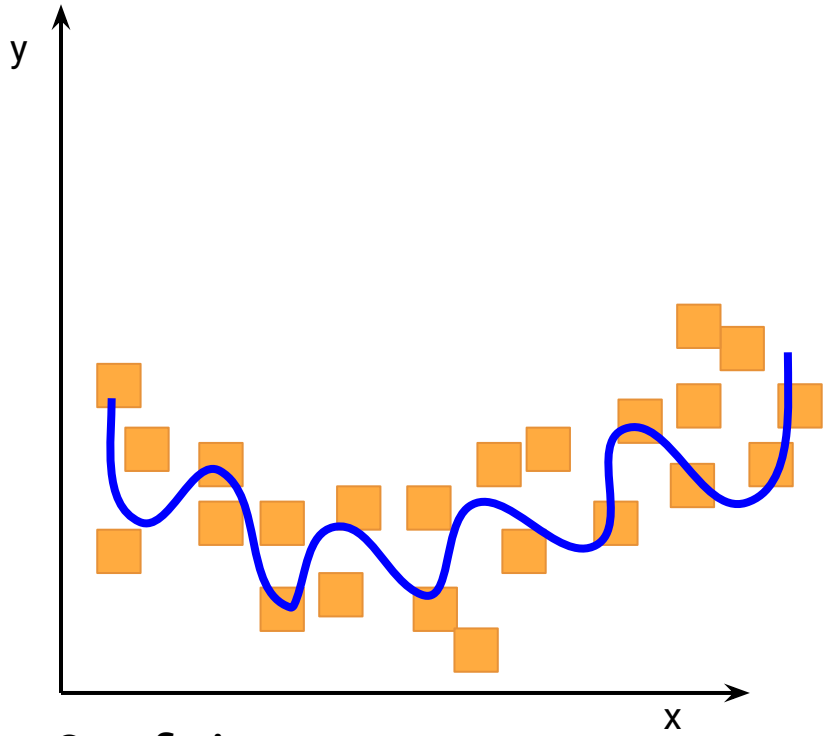


There are two major pain points that we will commonly encounter: overfitting and underfitting.

Overfitting

A model has **high variance**, or it adjusts too much to fit the training data and will not generalize well.

As a reminder, this is known as **overfitting** the model.



Overfitting

Overfitting

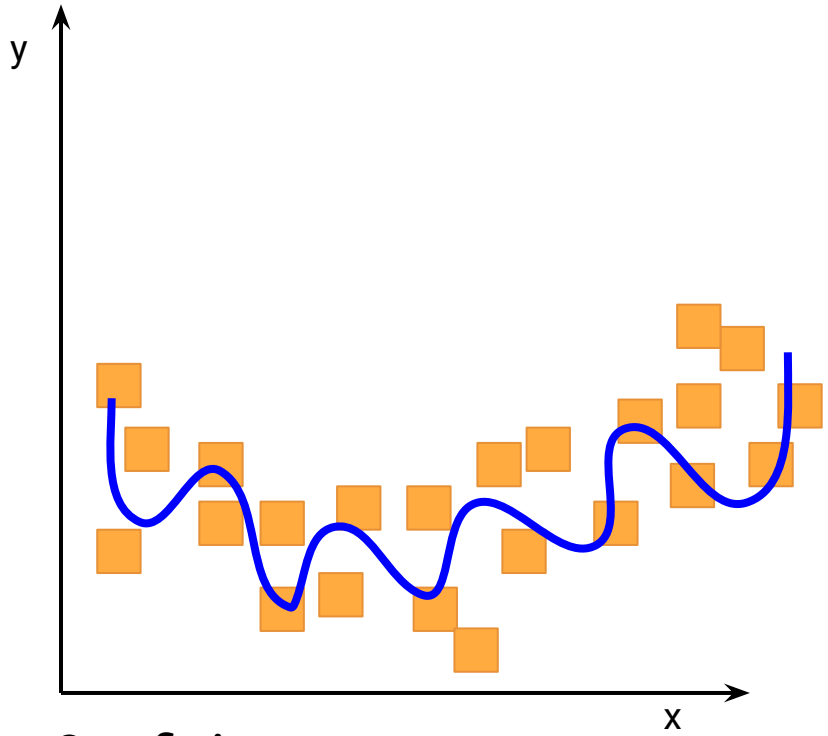
When a model is overfitting and does not meet performance expectations, it is usually due to one of two causes:

01

Training and test data are unbalanced, and the training data is not representative of the test data.

02

There is not enough complexity in the training data, and the model converges too quickly



Overfitting

To fix a model that has high variance and is overfitting, the most straightforward solution is to add more training data.

Add More Training Data

The first way to increase the training data is to collect more data for your input dataset.

Pro

This is the safest means of increasing training data if collected properly, using the same protocol as the initial data collection.

Con

The problem with collecting more input data is that it may be logistically or financially impossible.

Add more Training Data

The second means of increasing the training data is to change the split ratio of the training and testing data split of the original input data.

Pro

The benefit to this method is that no new data collection is necessary; therefore, there is no additional financial or logistical cost.

Con

There will be less data to use to test and validate the model. This means there is a higher risk that we might erroneously consider an underperforming model to be adequate.

Add More Training Data

Alternatively, we can keep our training data the same and retrain the model using less epochs.

Pro

This can be a safer alternative with smaller, simpler datasets

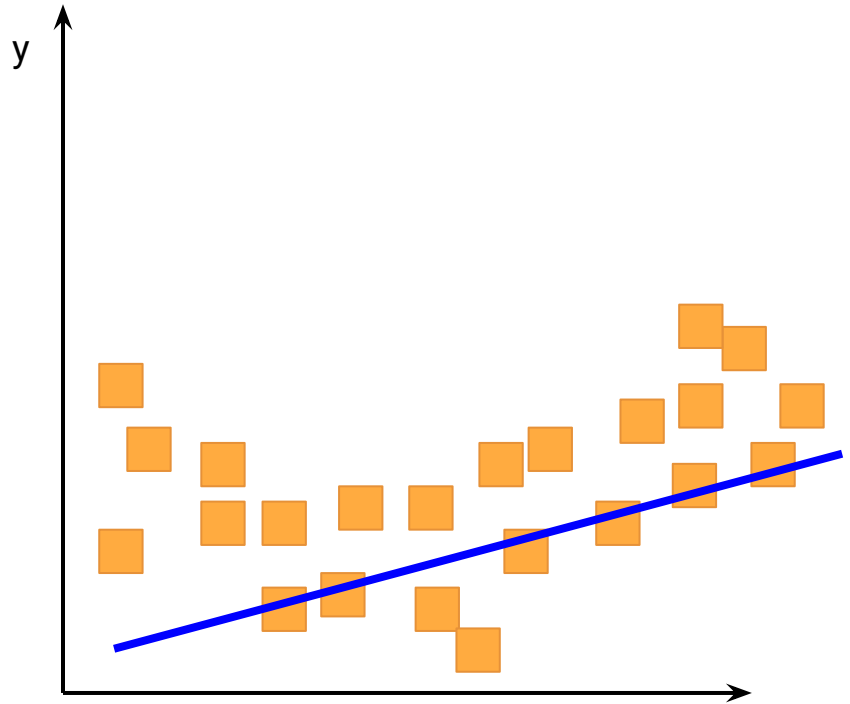
Con

It may be ineffective for larger datasets with many features.

Underfitting

A model can have **high bias**, or the input data is very noisy and the model is **underfitting** the data.

In other words, our model struggles to classify or predict our training dataset.



underfitting

Underfitting

When a neural network model is underfitting and does not meet performance expectations, it is usually due to one of two causes:

01

Training data that contains too many outliers or confusing variables

02

There are inadequate or inappropriate model design parameters, often referred to as **hyperparameters**.

Instructor Demonstration

Synaptic Boost

Although neural networks are tolerant of noisy characteristics in a dataset, neural networks can learn bad habits—just like our brains.

It is important to identify variables that contain a number of potential outliers because they can affect our data preprocessing, causing more important variables and features to disappear.

Hyperparameters

Hyperparameters that can be altered to achieve desired performance:

The number of neurons in a hidden layer

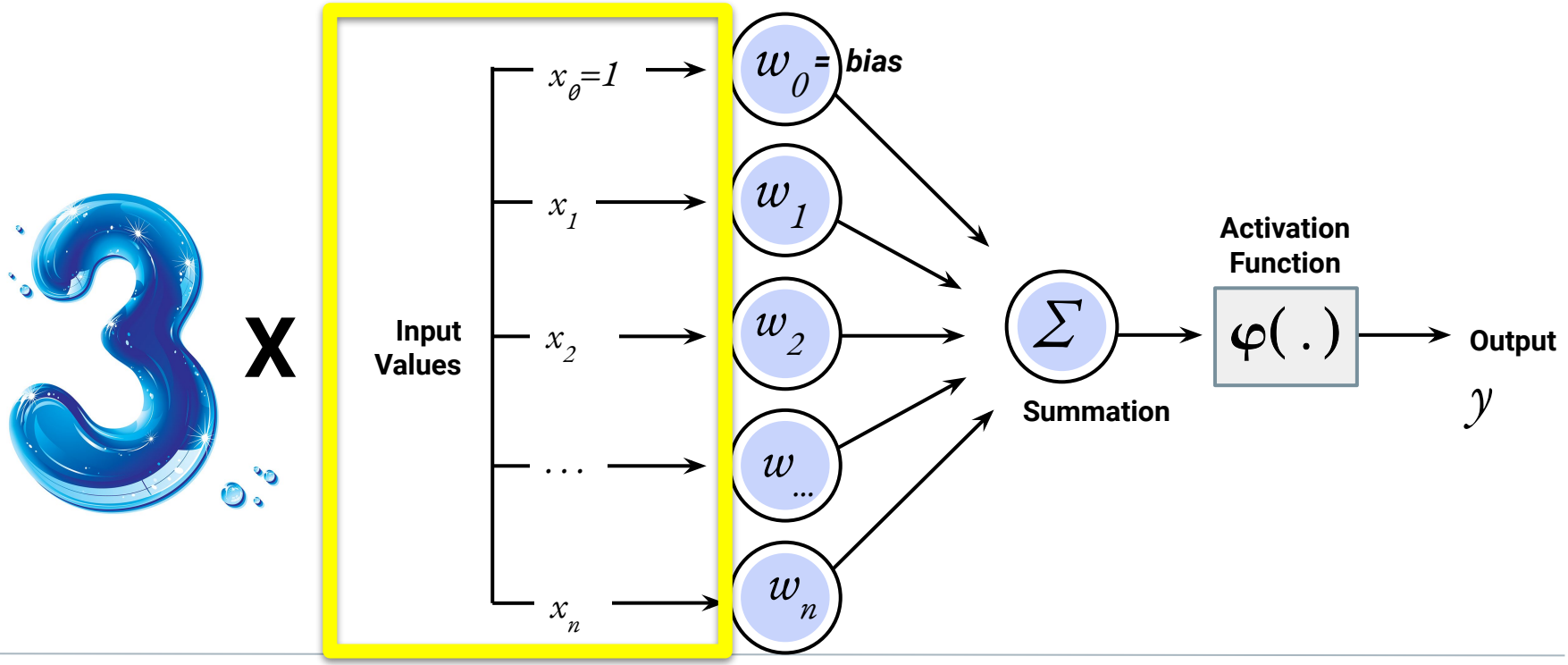
The number of hidden layers in a deep learning model

The activation function for each hidden layer

The number of epochs in the training regimen

Hyperparameters

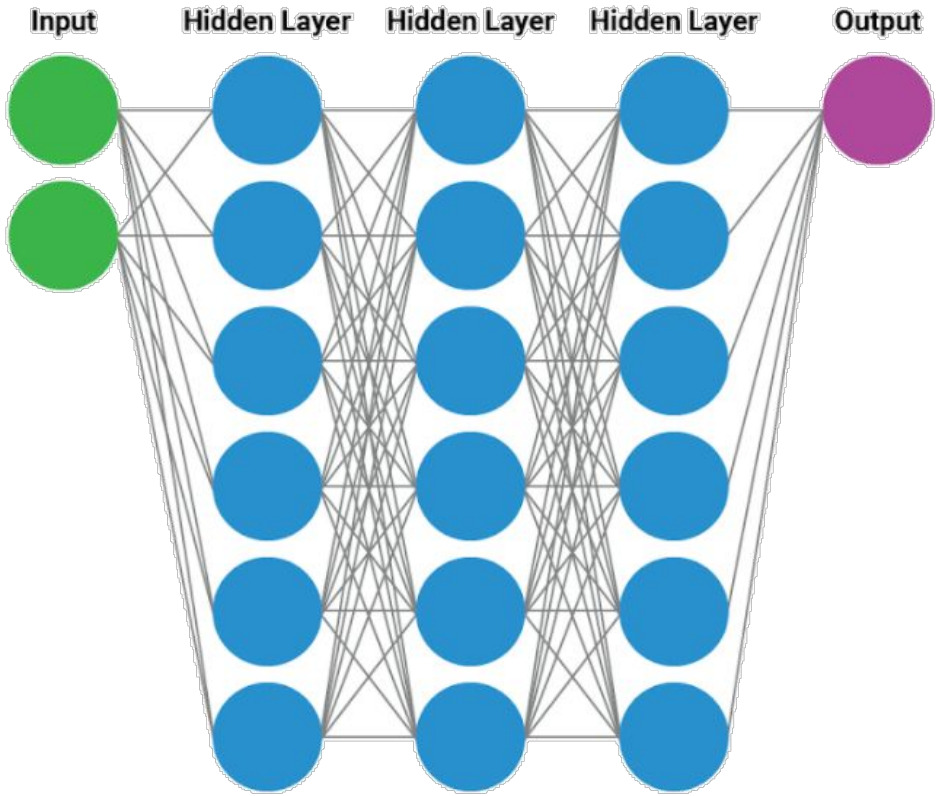
A good rule of thumb when building the initial model is to use two to three times as many neurons as there are input features, or values.



Hyperparameters

Similarly, we can try to boost the performance of a deep learning model by creating additional hidden layers.

Deep learning models require substantially more training iterations and memory resources with each additional hidden layer.

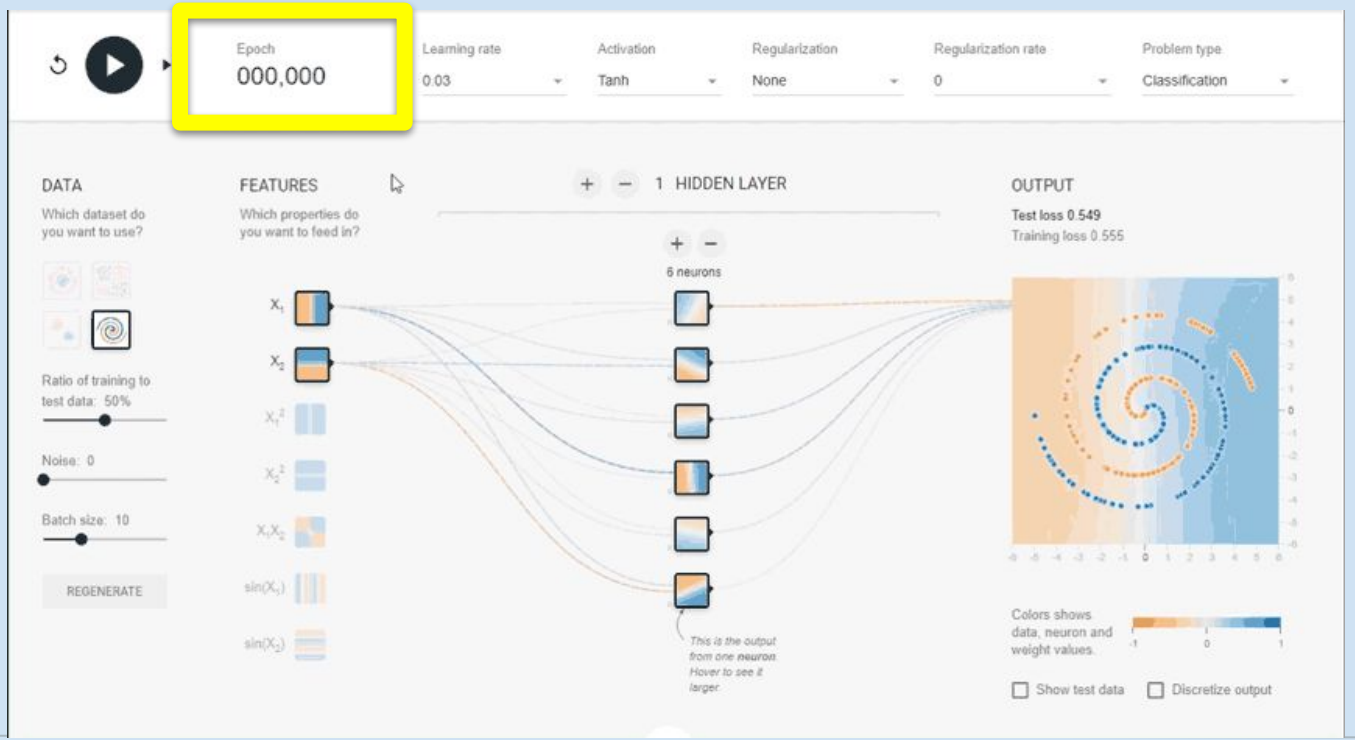


Depending on the shape and dimensionality of the input data, one activation function may focus on specific characteristics of the input values, while another activation function may focus on others.

A good rule of thumb is to try selecting slightly more complex activation functions for your hidden layers than your output layer.

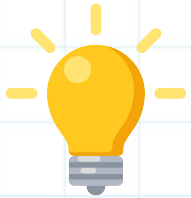
Epochs

If the model does not meet performance expectations after changing activation functions for each hidden layer, then we can increase the number of training epochs.



Epochs

As the number of epochs increases, so does the amount of information provided to each neuron.



Each training iteration tweaks the neuron's weight coefficients; therefore, each epoch increases the likelihood that the model is utilizing effective weight coefficients.

If the model produces weight coefficients that are too effective at analyzing the training data—the model is tailored to meet the demands of the current data—then it may not generalize well.

A good rule of thumb is to start with a smaller number of epochs, such as 100, and add more training epochs until training loss starts to decrease at a slower rate.

Hyperparameter Adjustment

Hyperparameter Adjustment	Pros 	Cons 
Add more neurons	Speeds up model, may reduce loss	More computational resources required
Add more layers	Considers more interactions between variables	More computational resources required
Change activation function	Drastically changes how a model interprets inputs	Not all new interpretations are effective
Add more epochs	Increases the likelihood that the model will achieve optimal weight coefficients	Increased risk of overfitting

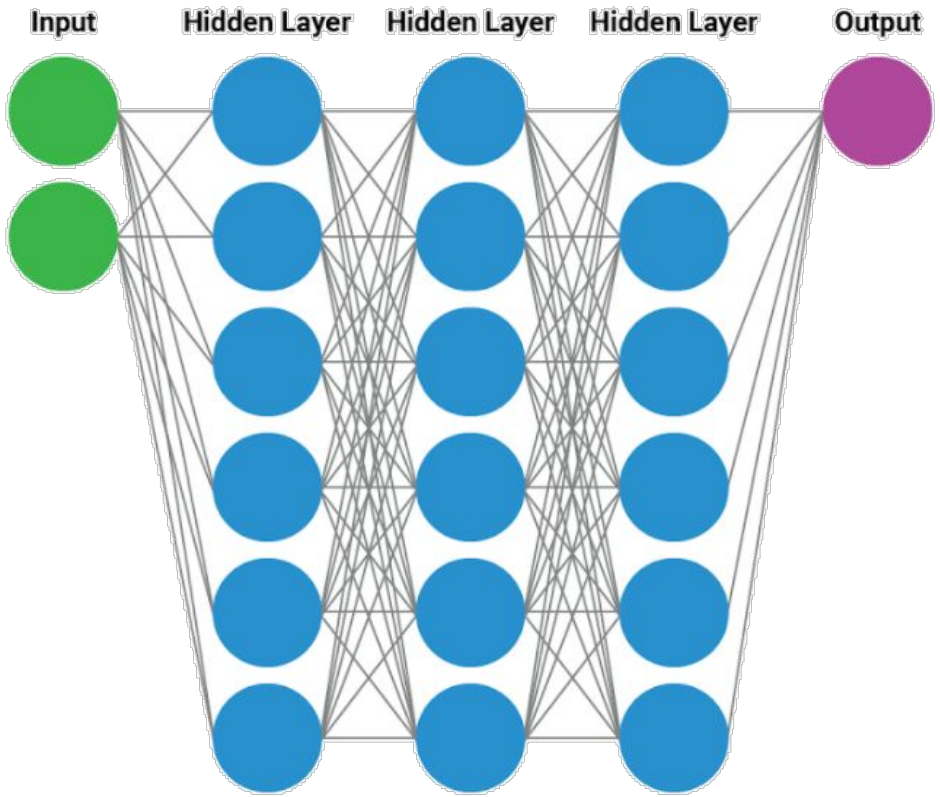
Questions?

Take the Guesswork out of Model Optimization

Model optimization is often the most tedious and critical step in designing an effective machine learning model.

Model Optimization

When it comes to neural network models, even small changes to model hyperparameters can cause large changes to overall model performance.



Instructor Demonstration

Model Optimization

Activity: Giving Your Model Building a Tune-up

In this activity, you will use `keras-tuner` to create a model that can adequately predict scikit-learn's `make_circles` dataset.

Suggested Time:

15 minutes

Time’s Up! Let’s Review.

Using Real-world Data in Keras Neural Network Models

Neural network models tend to require the most preprocessing of input data compared to all other statistical and machine learning models.

Real-world Neural Network

If a bank wanted to build a neural network model to identify if a company was eligible for a loan, it might look at factors such as a company's net worth.

If the bank's input dataset contained information from large fortune 500 companies, such as Google and Facebook, as well as small mom-and-pop stores, the variability in net worth would be outrageous.

Without normalizing the input data, a neural network could look at net worth as being a strong indicator of loan eligibility, and as a result, it could ignore all other factors, such as debt-to-income ratio, credit status, or requested loan amount.

Instead, if the net worth was normalized on a factor such as number of employees, the neural network would be more likely to weigh other factors more evenly against net worth.

This would result in a neural network model that assesses loan eligibility more fairly without introducing any additional risk.

Preprocessing the input data–

we alter the input dataset before any computational model training or evaluation.

Preprocessing the Input Data

Consider the following `eye_color` variable containing a list of eye colors from different people.

Eye_Color
Blue
Brown
Brown
Brown
Hazel
Green
Hazel
Brown
Blue
Brown

Preprocessing the Input Data

Each row has only one column with a value of 1—the corresponding categorical variable from the original dataset.

Eye_Color:Blue	Eye_Color:Brown	Eye_Color:Hazel	Eye_Color:Green
1	0	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	1	0	0

This binary encoding ensures that each neuron receives the same amount of information from the categorical variable.

The process of reducing the number of unique categorical values in a dataset is known as **bucketing**, or **binning**.

Bucketing

There are two approaches to bucketing categorical data:

01

Collapse all of the infrequent and rare categorical values into a single “other” category.

This approach takes advantage of the fact that uncommon categories and "edge cases" are rarely statistically significant.

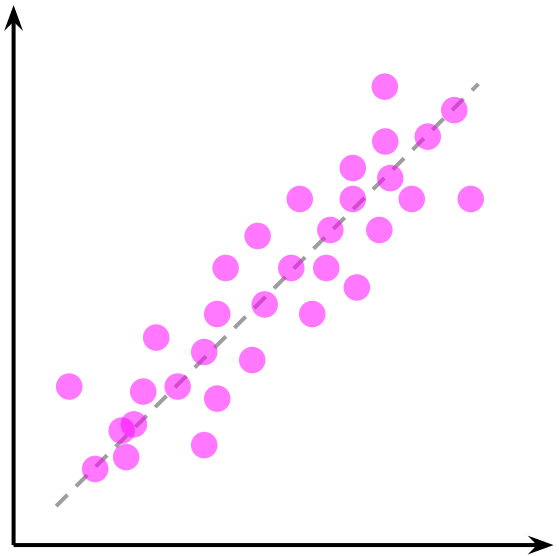
02

Create generalized categorical values and reassign all data points to the new corresponding values.

This approach collapses the number of unique categorical values and maintains relative order and magnitude.

Bucketing

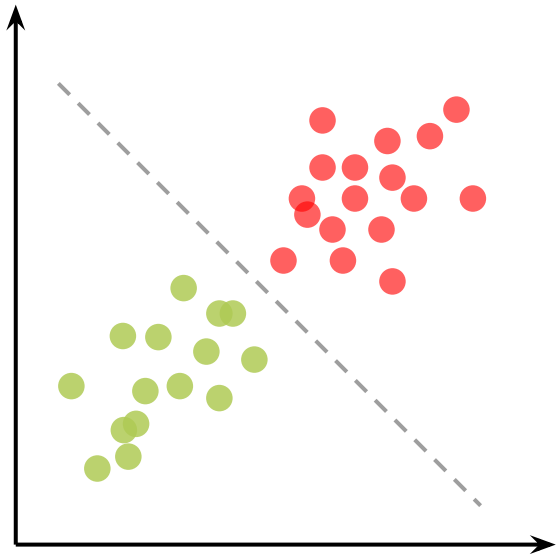
Regression and classification models are unlikely to be able to use rare categorical values to produce robust models.



Regression

Patients

VS.



Classification

Healthy Weight

Overweight

A good rule of thumb is to only apply a bucketing strategy when the categorical variables contain 10 or more unique values.

Preprocessing the Input Data

Reasons why we should not train a neural network model on raw numerical data:

01

Raw data often has outliers or extreme values that can artificially inflate a variable's importance.

02

Numerical data can be measured using different units across a dataset—such as time versus temperature or length versus volume.

03

The distribution of a variable can be skewed, leading to misinterpretation of the central tendency.

What module should we use to standardize the numerical data?

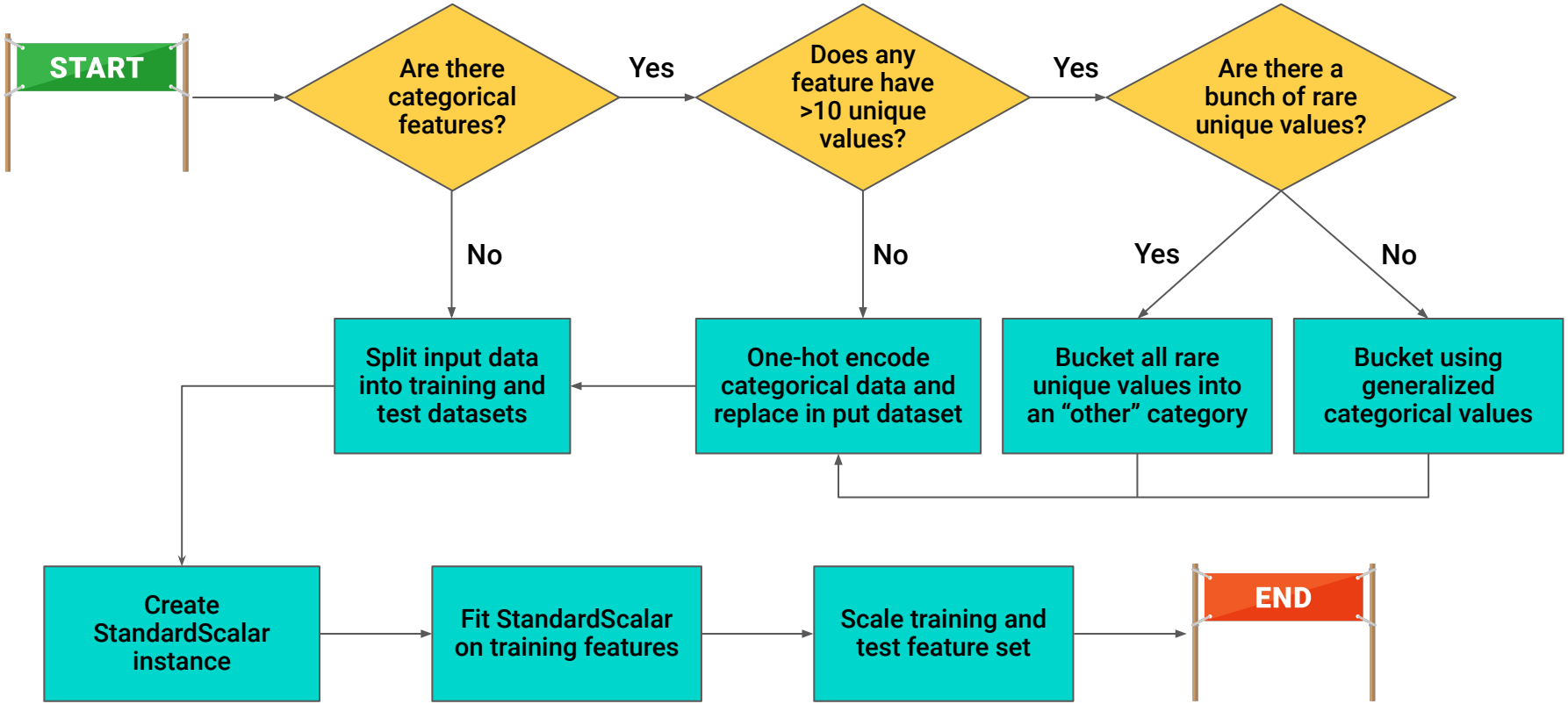
What module should we use to standardize the numerical data?

We standardize the numerical data using scikit-learn's `StandardScaler()` module.

Instructor Demonstration

Getting Real with Neural Network Datasets

Preprocessing Neural Network Input Data



Activity: Detecting Diabetes through Deep Learning

In this activity, you will preprocess a medical dataset and create a deep learning model capable of predicting whether a patient will be diagnosed with diabetes

Suggested Time:

15 minutes

Questions?