Deep Neural Networks

Fintech

Lesson 13.2

Class Objectives

By the end of the class, you will be able to:

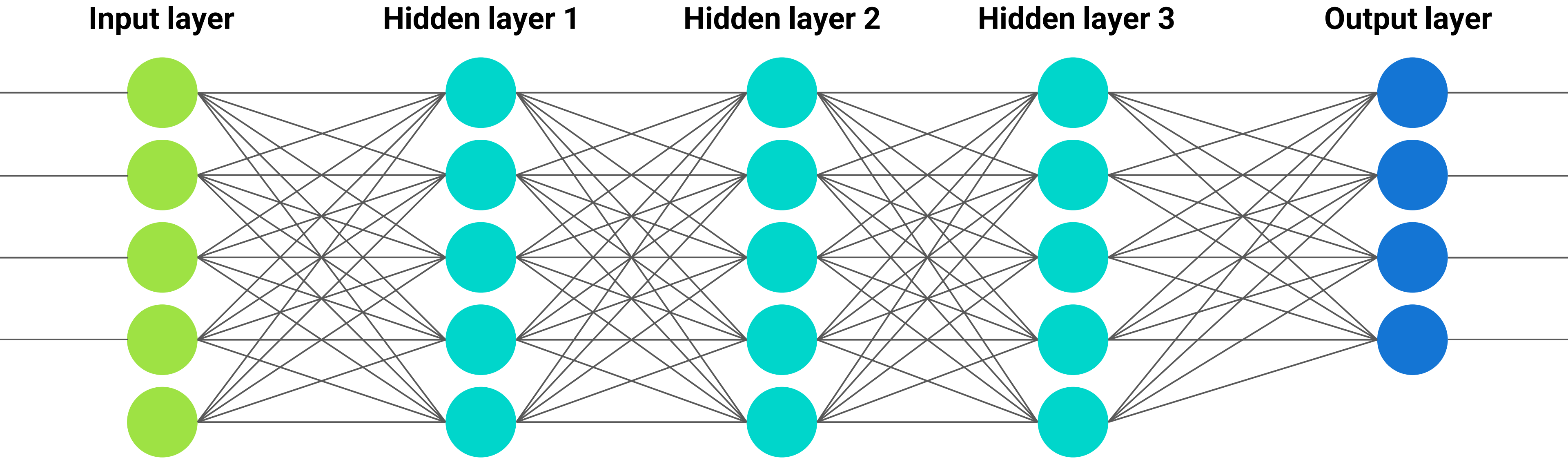
- Explain the difference between neural networks and deep neural networks.
- Build deep learning models by using Keras.

WELCOME

Today, we will dive into constructing deep learning models with real-world data.

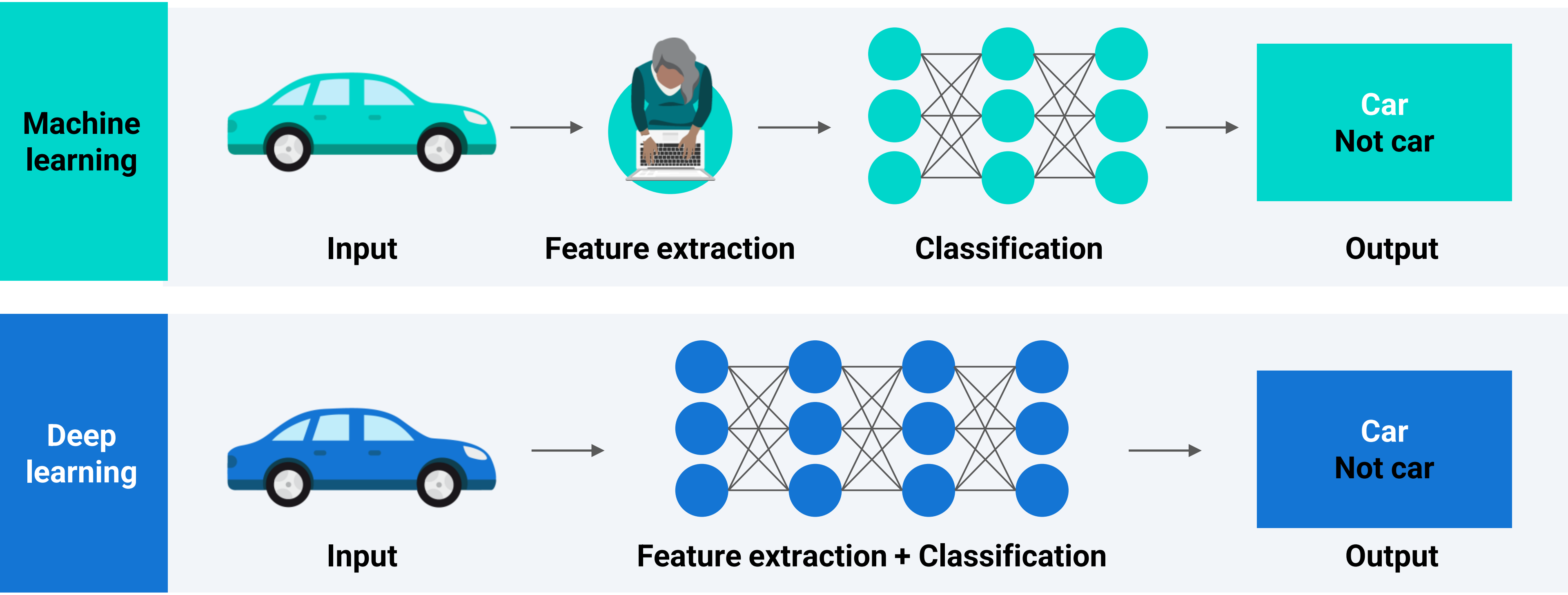
Deep Neural Networks

Generally speaking, deep learning models are neural networks with more than one hidden layer.



Machine Learning vs. Deep Learning

Deep neural networks are much more effective than traditional machine-learning approaches at discovering nonlinear relationships among data.



Deep Neural Networks

Deep neural networks are often the best choice for complex or unstructured data like:

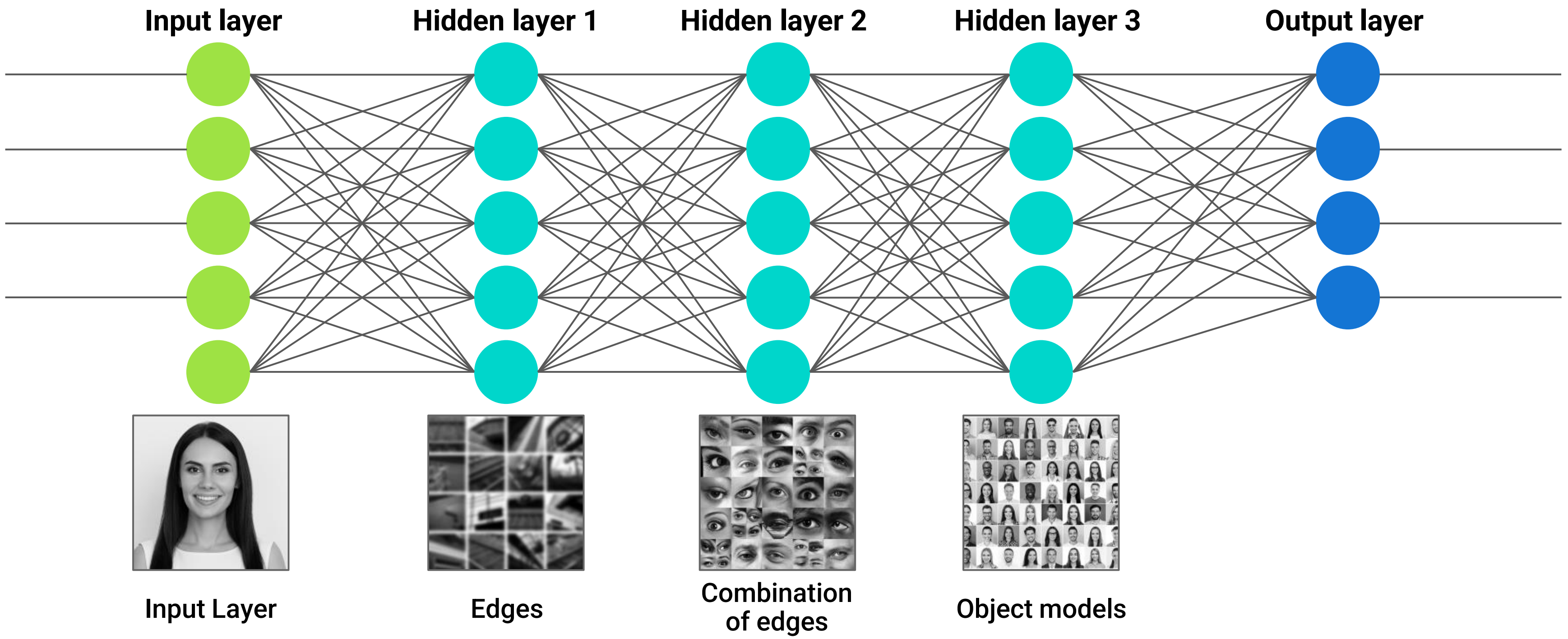
Image

Text

Voice

Deep Neural Networks

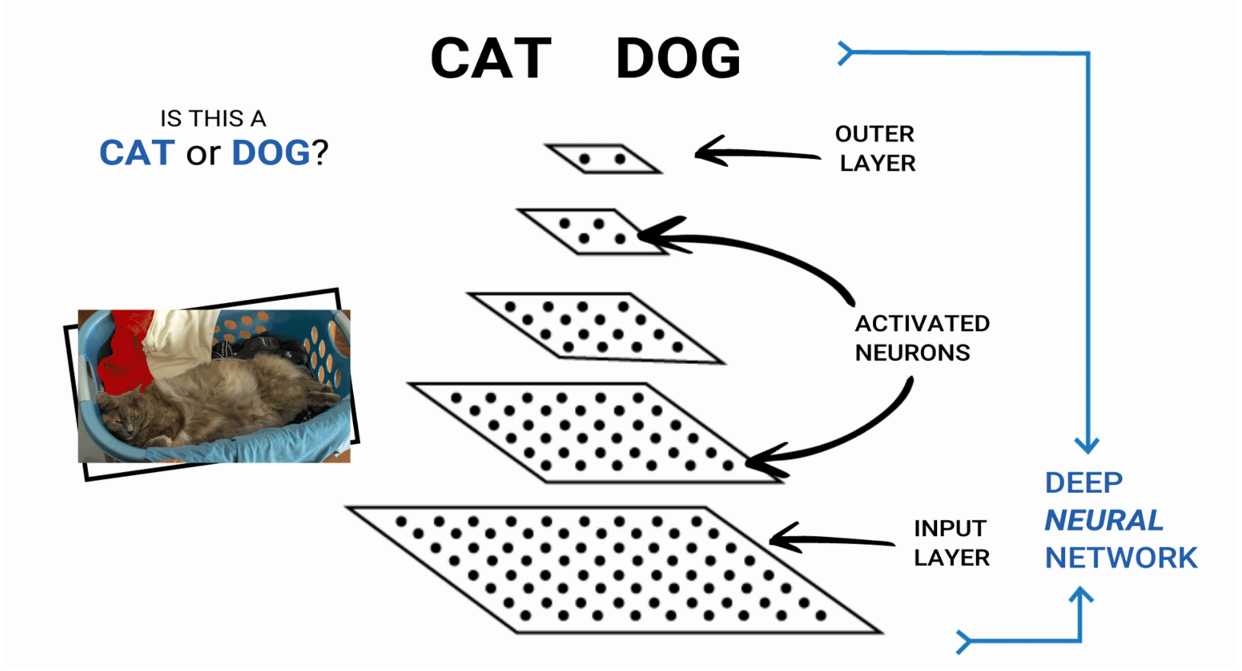
In image recognition, each layer can identify different image features in the process of defining or identifying the image.



Introduction to Deep Learning

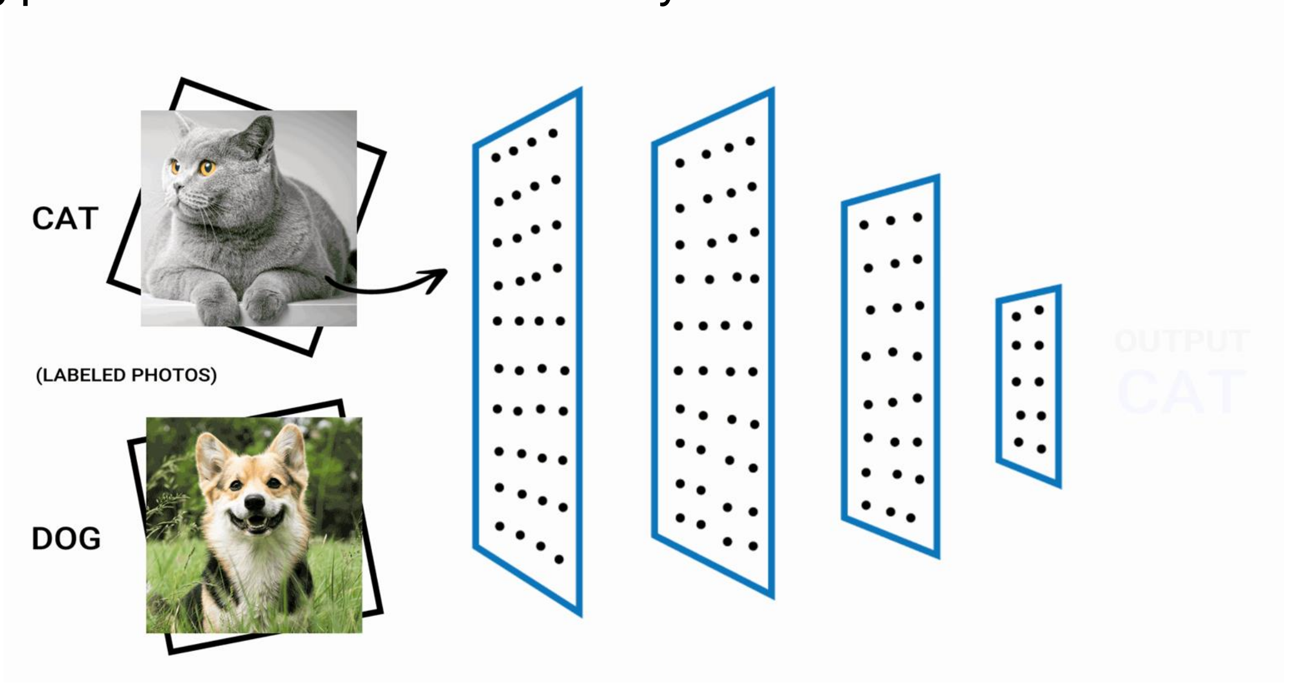
Introduction to Deep Learning

Neural networks calculate the weights of various input data and pass them to the next layer of neurons. This process continues until the data reaches the output layer, which makes the final decision on the predicted category or numerical value of an instance.



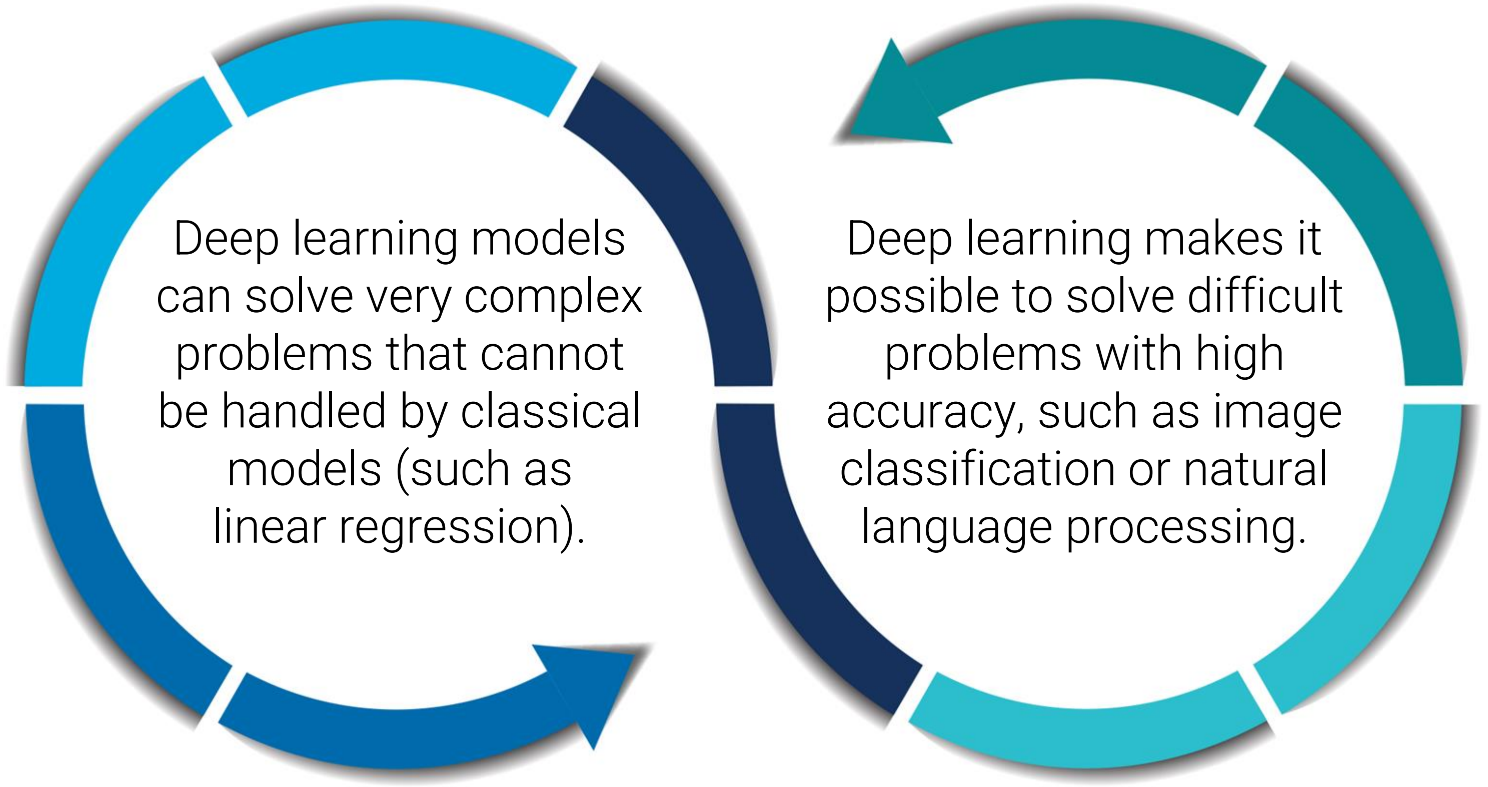
Introduction to Deep Learning

While definitions vary, we can consider neural networks with more than one hidden layer to be deep learning models. The decreasing cost and greater availability of computing power has increased our ability to create and use these models.



Each additional layer of neurons makes it possible to model more complex relationships and concepts.

Introduction to Deep Learning



Deep learning models can solve very complex problems that cannot be handled by classical models (such as linear regression).

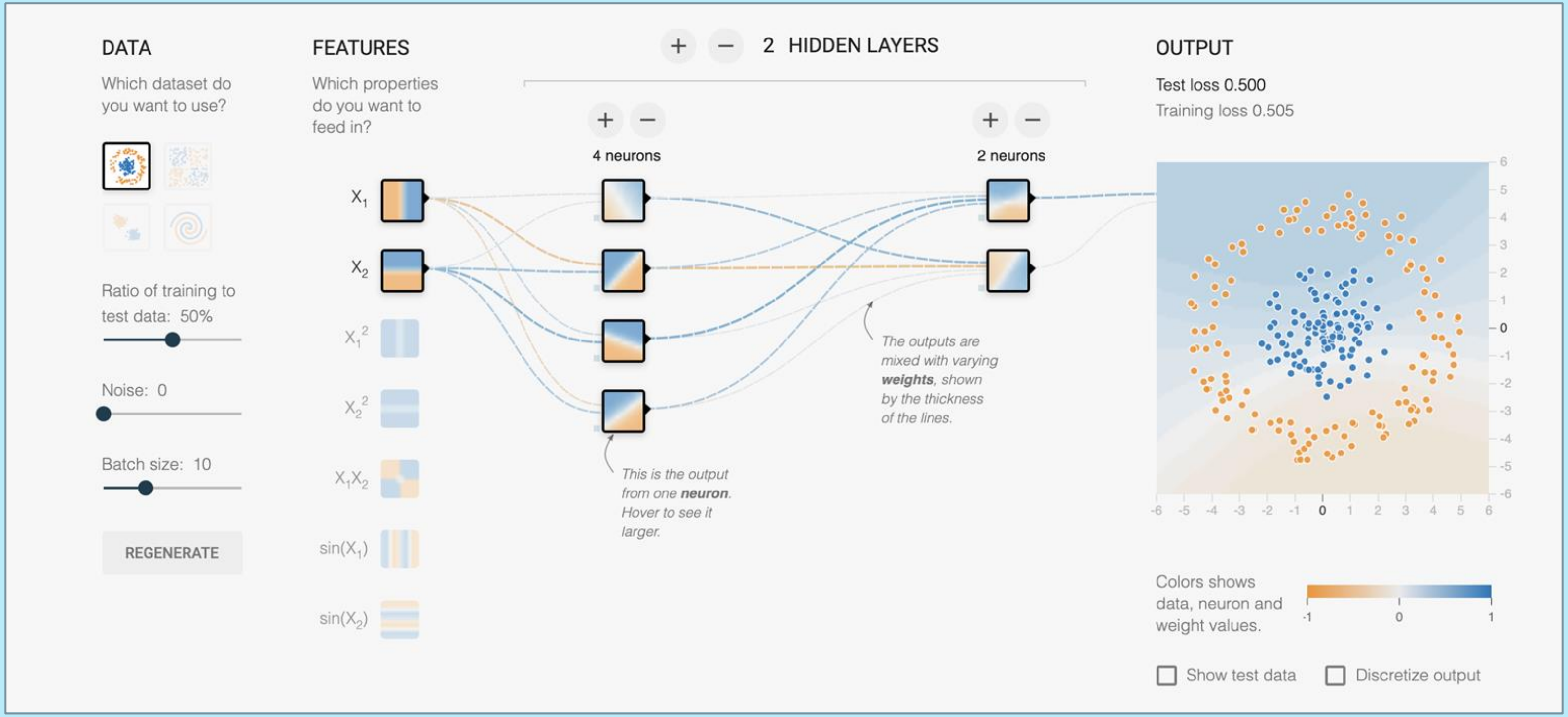
Deep learning makes it possible to solve difficult problems with high accuracy, such as image classification or natural language processing.

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 blac 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.


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
+ − 2 HIDDEN LAYERS
4 neurons
2 neurons
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.
OUTPUT
Test loss 0.500
Training loss 0.505
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

TensorFlow Playground

Regardless of what machine learning model or technology is used, the same general modeling workflow exists across all data science:

01 Decide on a model and create a model instance.

02 Split into training and testing sets and preprocess the data.

03 Train/fit the training data to the model.

04 Evaluate the model for predictions and transformations.

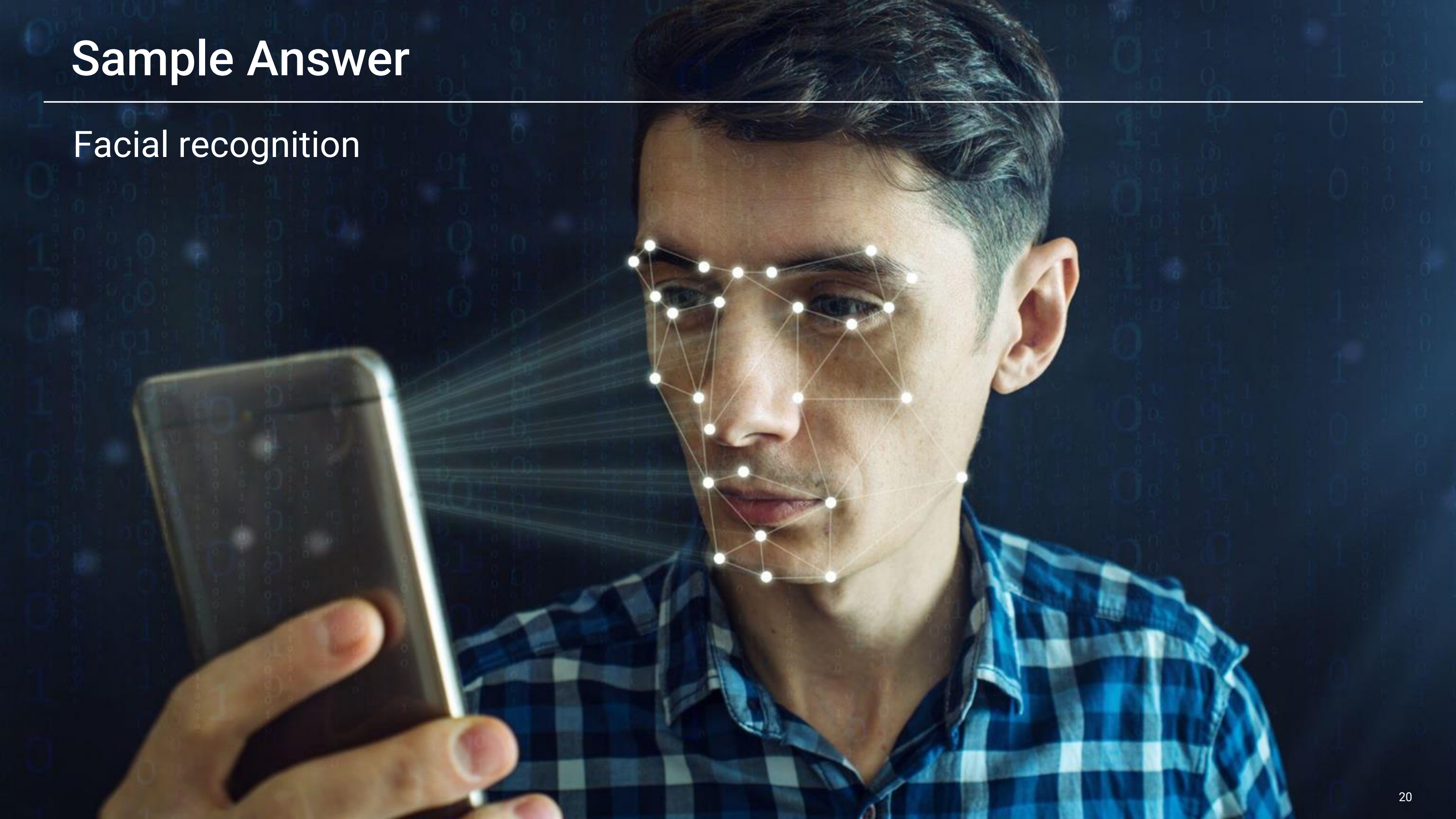
Instructor Demonstration

TensorFlow Playground

What are some examples of classification or regression problems that might benefit from a deep learning model?

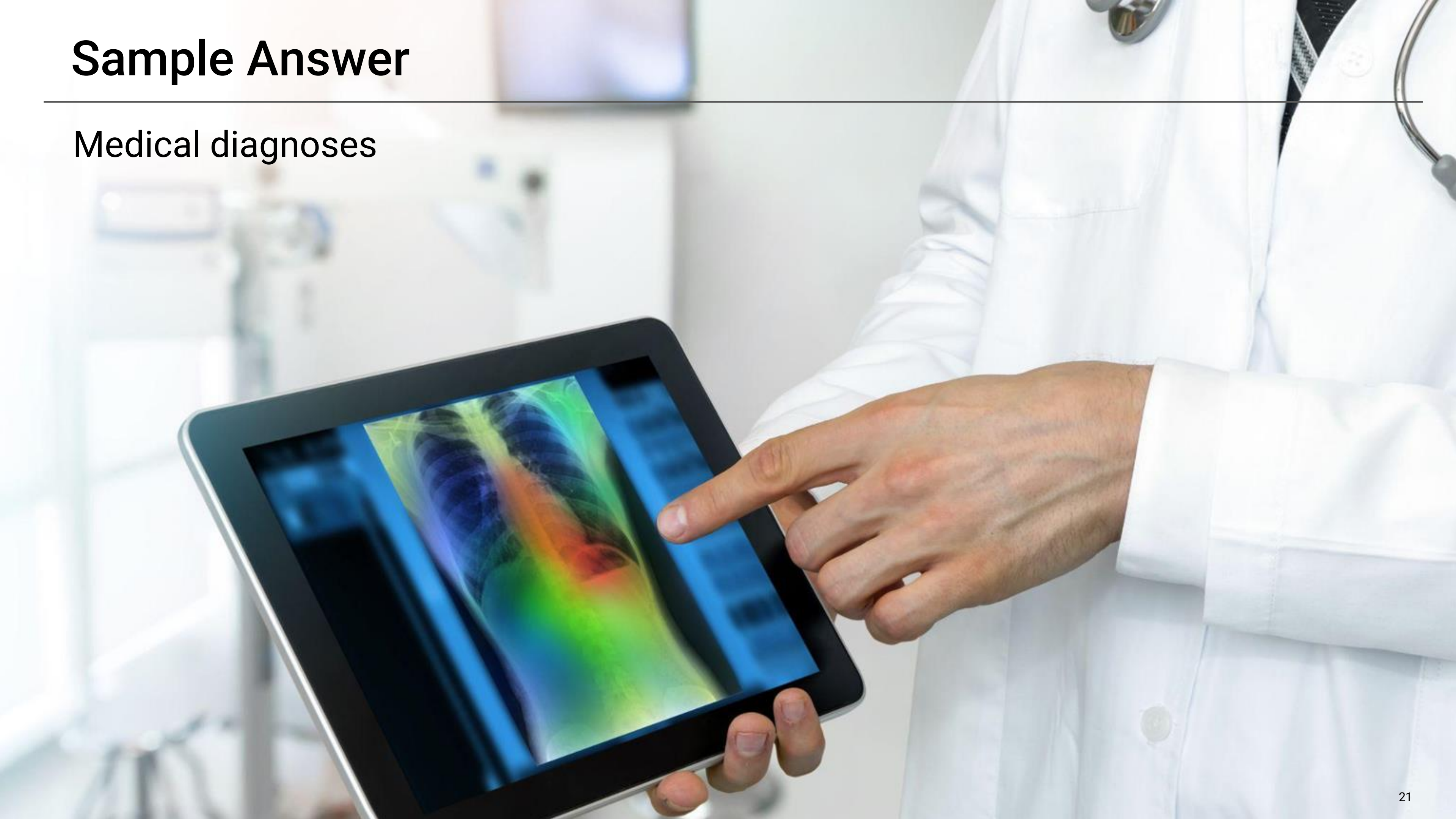
Sample Answer

Facial recognition



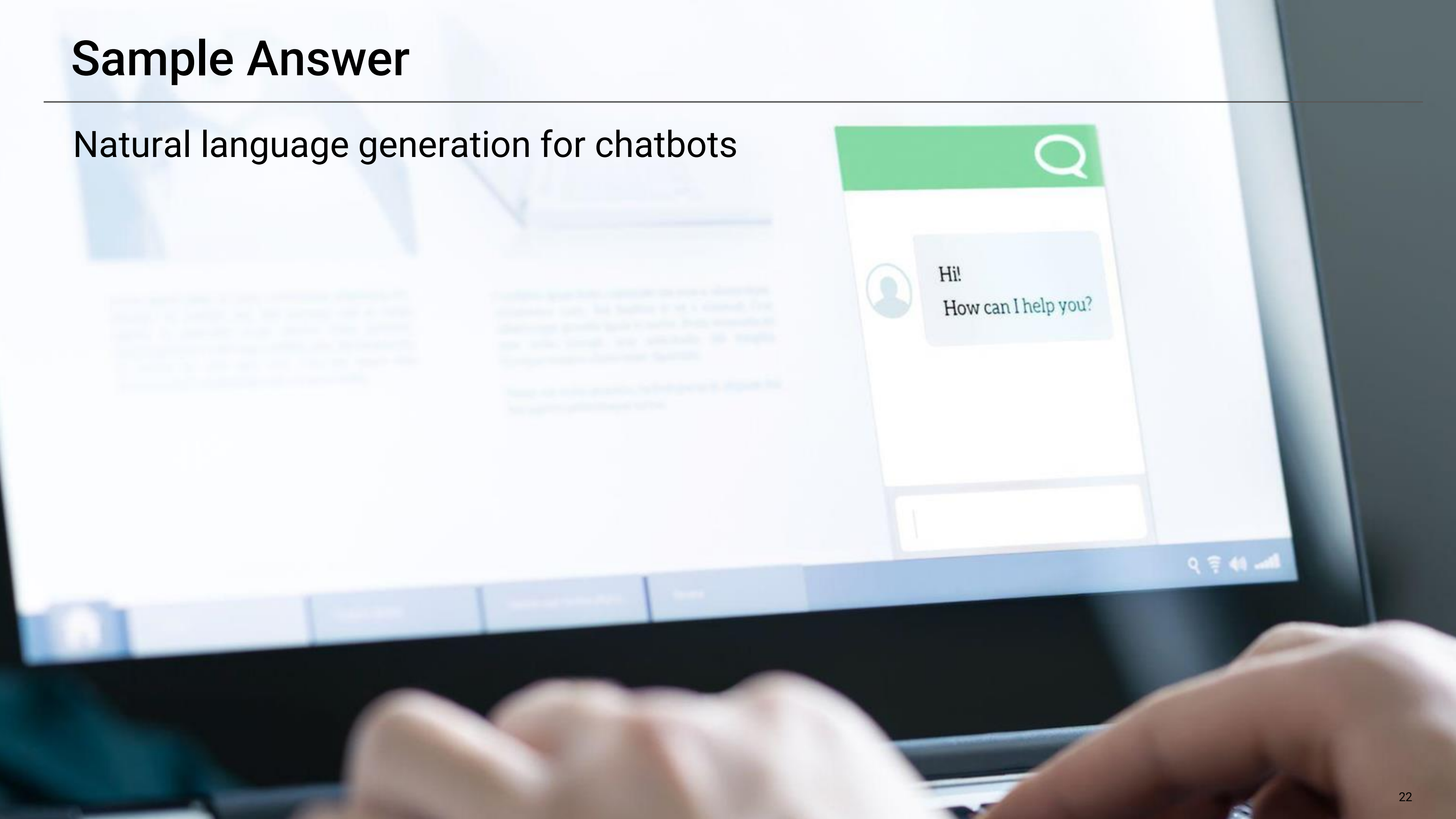
Sample Answer

Medical diagnoses



Sample Answer

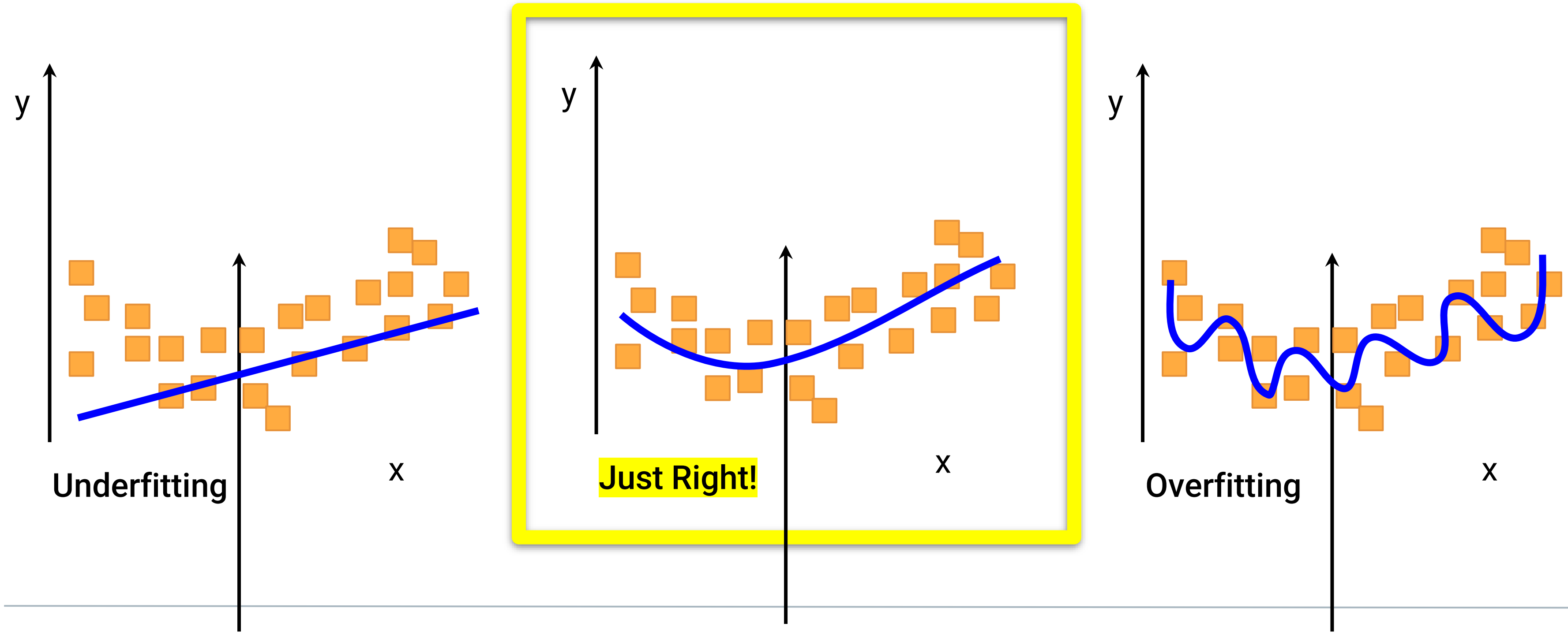
Natural language generation for chatbots



How can we determine if a neural network has too many layers of neurons?

Sample Answer:

We can watch for signs of overfitting. For example, test accuracy is far lower than training accuracy.



Questions?

Instructor Demonstration

Deep Learning with Keras

Questions?

Activity: Predicting Fraudulent Transactions

In this activity, you will will build a deep learning model that can predict financial fraud.

Suggested Time:

20 minutes

Time’s Up! Let’s Review.

Questions?

Countdown timer

15:00

(with alarm)

Time to Code

Deep Learning with Nonlinear Data

Suggested Time:

20 minutes

How can we improve the model accuracy?

Sample Answers

01

We can add more neurons to the hidden layer.

02

We can add a second hidden layer.

03

We can test with different activation functions.

How Can We Improve the Model Accuracy?

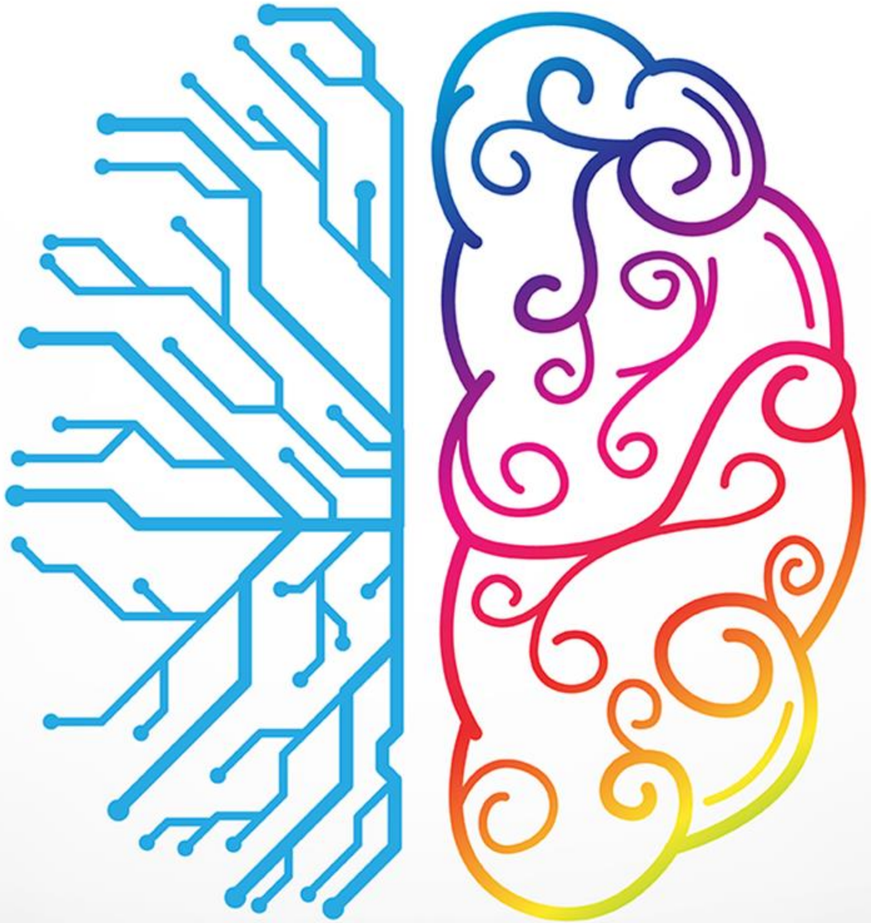
Sample Answers:

- Adding more neurons to the model is a possible solution, but we might overfit the model.
- We could add a second layer to the model. This is a good solution that is part of deep learning. The next class will cover this topic.
- We can test with different activation functions. This is typically the first thing to try, especially when dealing with nonlinear data.
- Using more epochs for training is another strategy to improve the model's accuracy.

Spend some time modifying the code to use a different number of hidden layer nodes, and observe how the model accuracy changes.

Modeling neural networks is a mix of science and art.

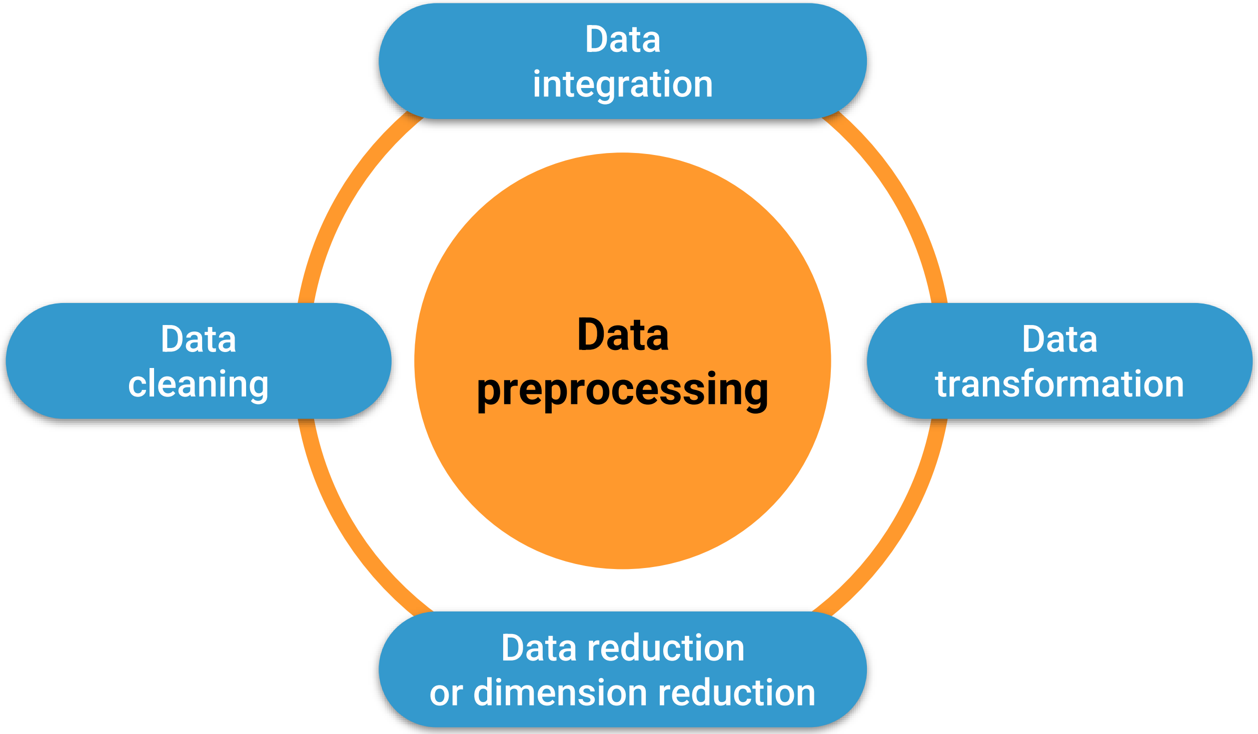
You'll find the best model through playing around with the number of neurons and testing different activation functions.



Encoding Categorical Variables

Encoding Categorical Variables

For all machine learning models (including neural networks), our first step is to preprocess the data.



Encoding Categorical Variables

Neural networks often have difficulty training on numerical variables that are represented in different units or that have different scales of magnitude.

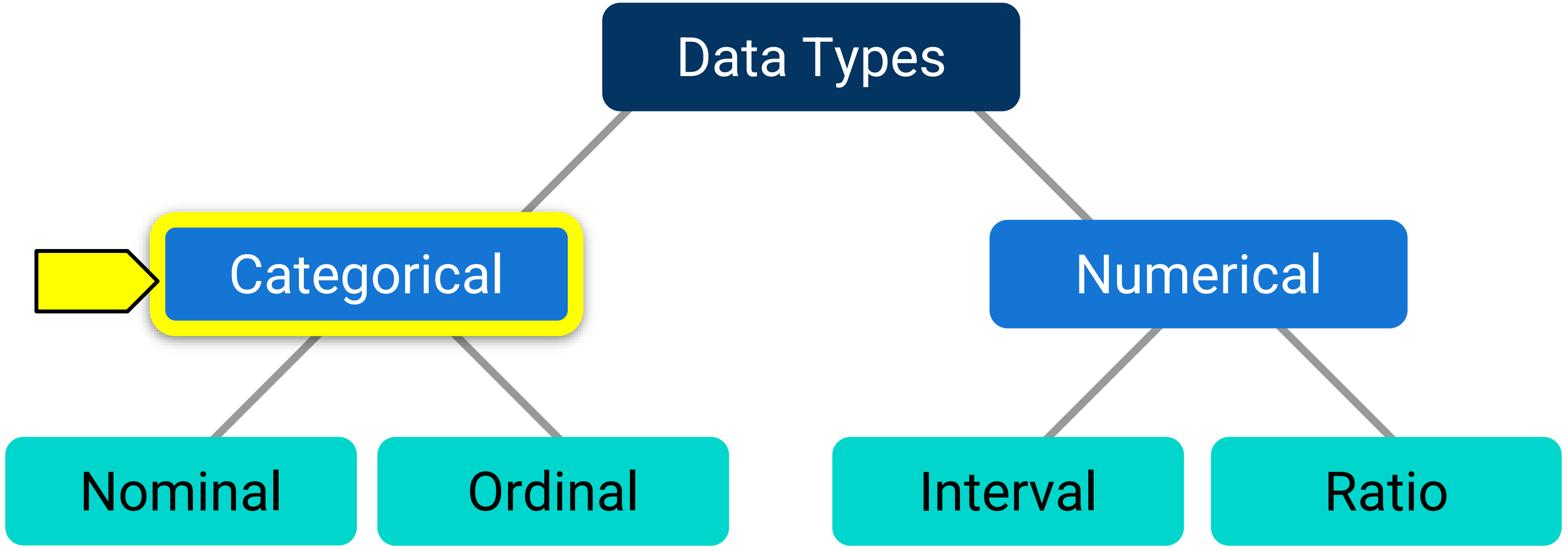
We previously used the `StandardScaler` module from scikit-learn to mitigate this problem for numerical variables.

This module scales the values of different variables by normalizing data to center around the mean.

Encoding Categorical Variables

Neural networks cannot process categorical variables in their raw forms.

As the name implies, categorical variables represent categories, like a person's city of residence, or different industries of companies in a stock portfolio, in their raw forms.



Encoding Categorical Variables

Remember…

We previously used the Pandas `get_dummies()` function to solve this problem for categorical variables.

However, `get_dummies` can run into trouble if the testing dataset contains categories that are not present in the training dataset.

To account for this potential `get_dummies` issue, we will use `OneHotEncoder`, a scikit-learn module that allows us to specify what happens when a new category appears in testing data.

One-Hot Encoding

One-hot encoding involves taking a categorical variable, such as color, and creating three new variables of the colors. Each instance of the data now shows a **1** if it corresponds to that color, and **0** if it does not.

	A	B	C	D	E	F	G	H	I
1	Original data:			One-hot encoding Format					
2	id	Color		id	White	Red	Black	Purple	Gold
3	1	White		1	1	0	0	0	0
4	2	Red		2	0	1	0	0	0
5	3	Black		3	0	0	1	0	0
6	4	Purple		4	0	0	0	1	0
7	5	Gold		5	0	0	0	0	1

Instructor Demonstration

Encoding Categorical Variables: OneHotEncoder

Questions?

Activity: Encoding Categorical Variables

In this activity, you will build a deep learning model that can predict employee attrition. In doing so, you'll use `onehotencoder` to help constructor your input data.

Suggested Time:

20 minutes

Time’s Up! Let’s Review.

Encoding Categorical Variables

Before going through the code, our work will be broken into three sections:

01 Preprocess the data.

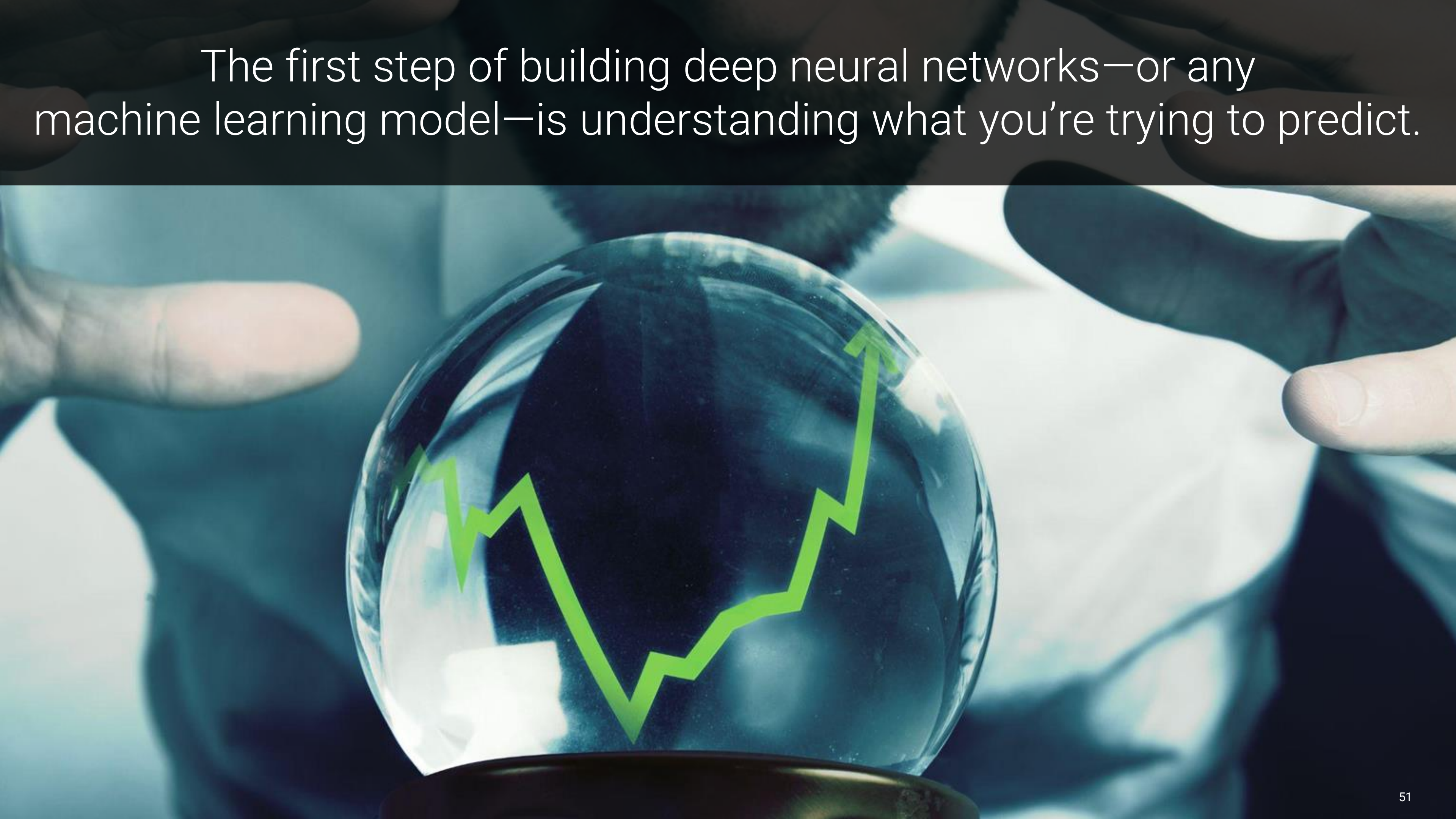
02 Create a neural network model to predict employee attrition.

03 Train and evaluate the neural network model.

Questions?

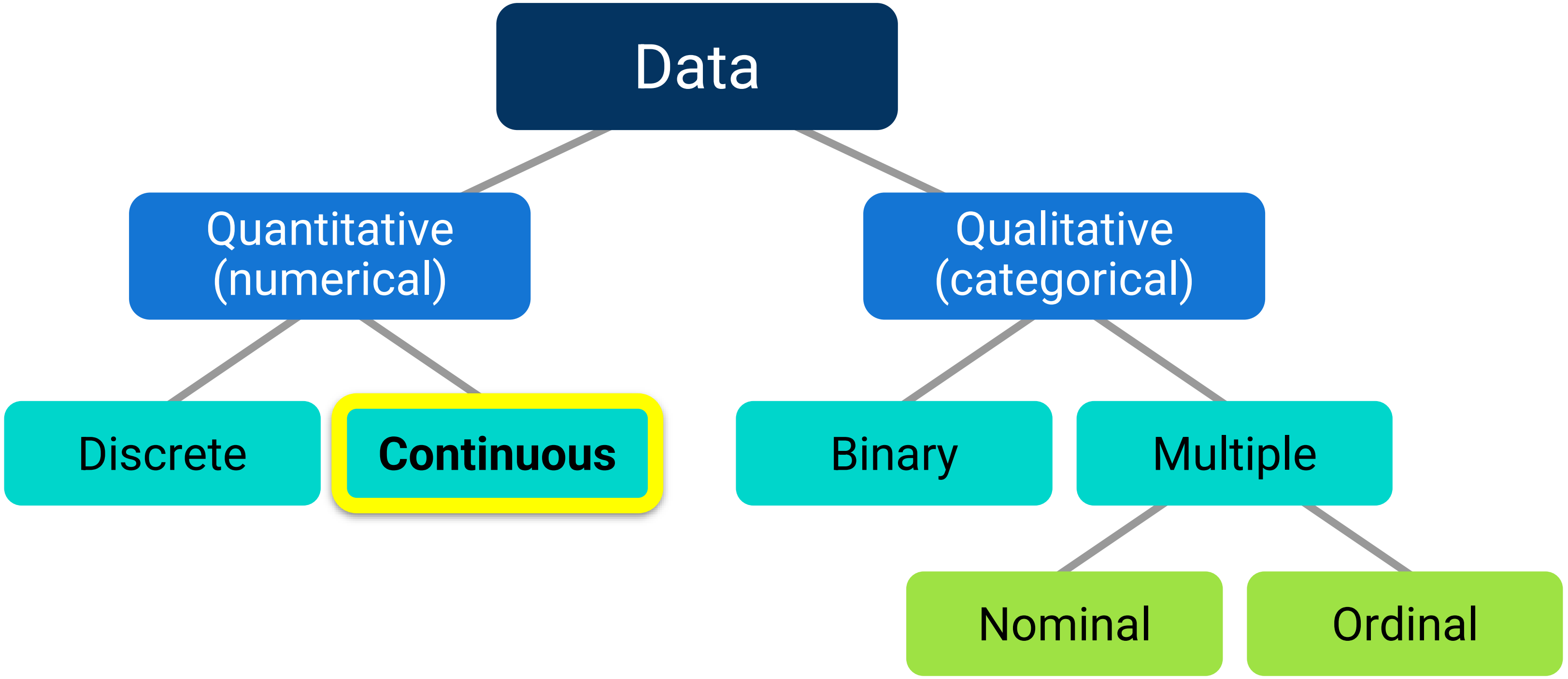
Multiclass Prediction in Deep Learning

The first step of building deep neural networks—or any machine learning model—is understanding what you’re trying to predict.



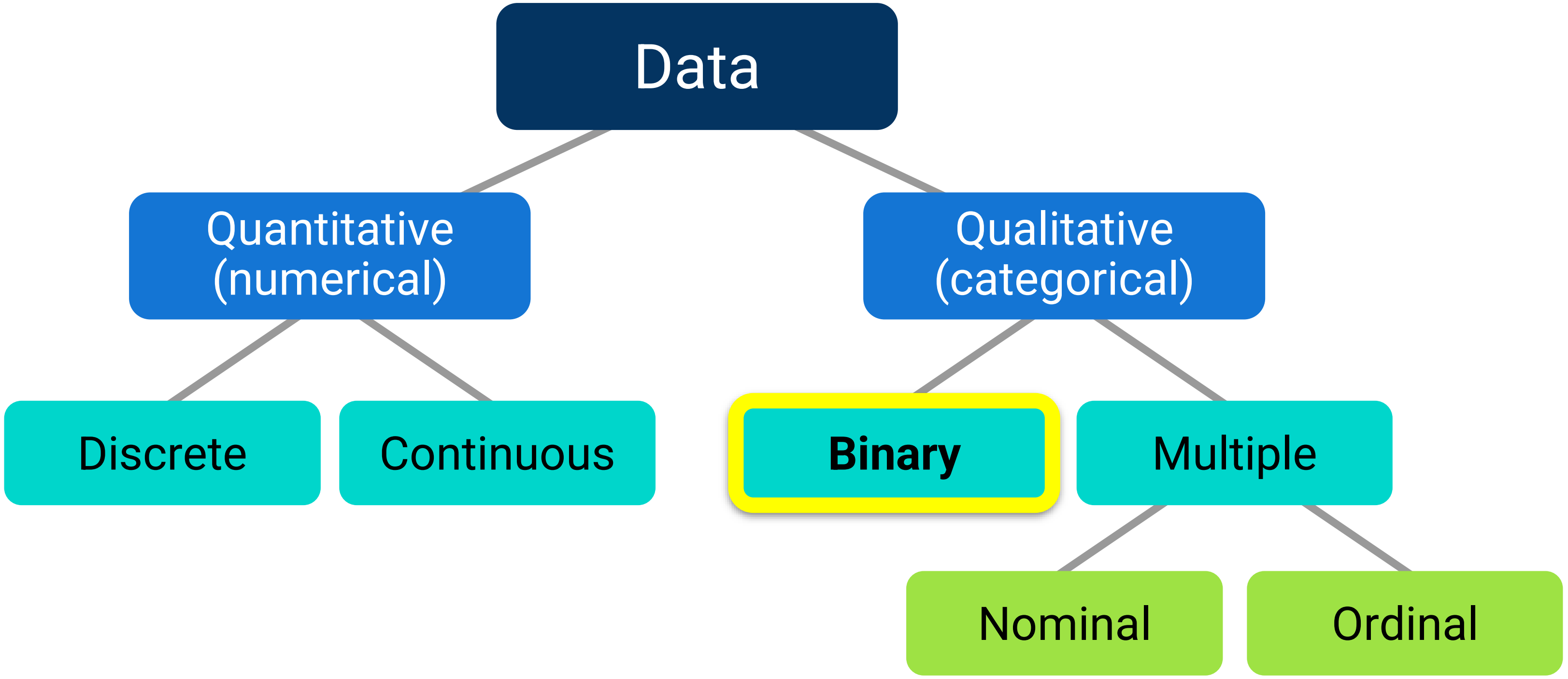
Continuous Variable

When y is something like a stock return, the growth rate of a company, or other wide-ranging continuous number, our target variable is continuous.



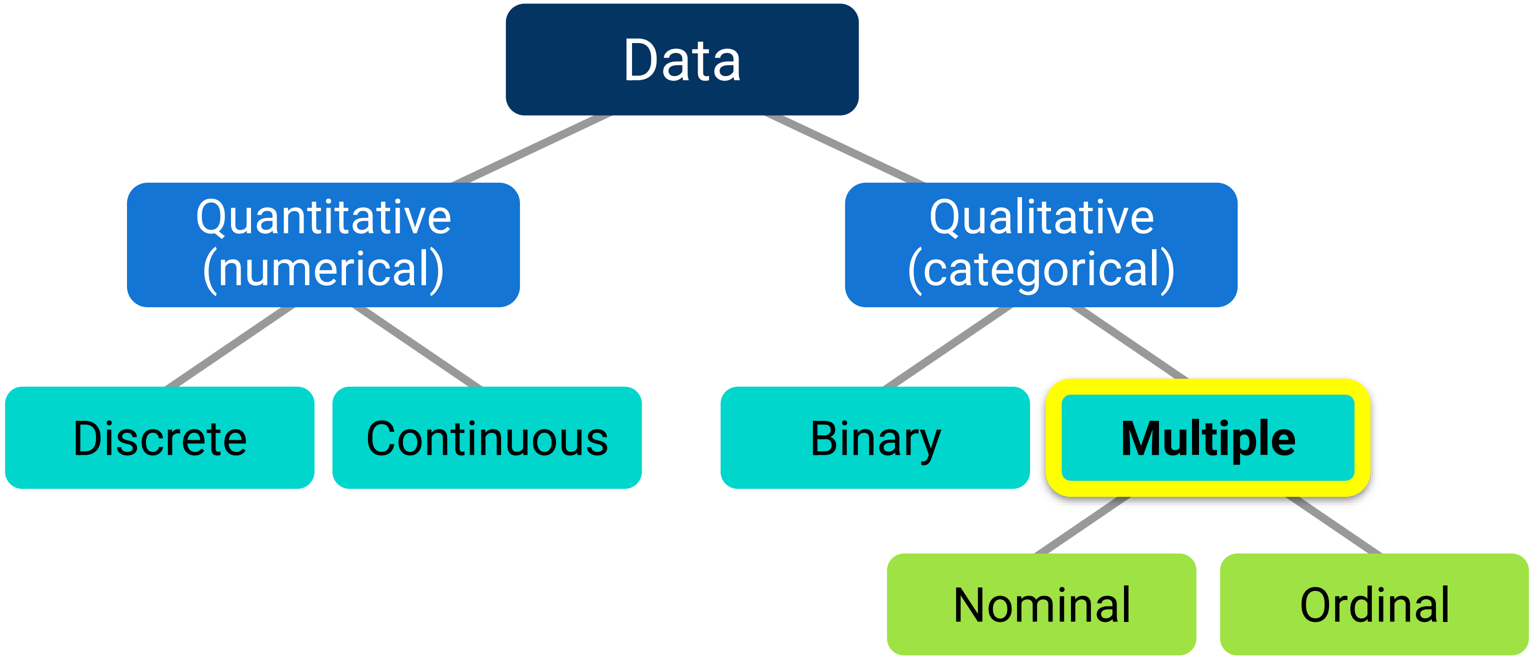
Categorical Variables: Binary

In finance, there are many “Yes or No” type events that we want to predict: they either will or won’t happen.

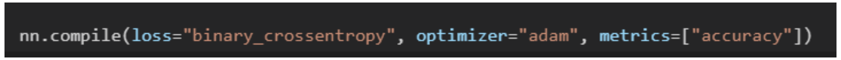


Categorical Variables: Multiclass

When y is a **multi-class categorical** variable, we're trying to predict among many outcomes. For example, we might try to predict the credit score ranking ("Bad," "Okay," "Good," "Great") to which a consumer belongs.



Adam Optimizer

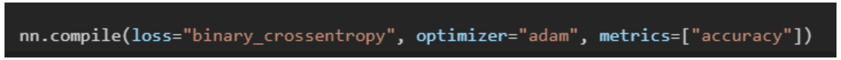


```
nn.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

- **Adam Optimizer:** An extension to Stochastic gradient decent and can be used in place of classical stochastic gradient descent to update network weights more efficiently.
- Author Diederik P. Kingma of OpenAI and Jimmy Lei Ba of University of Toronto
    - Benefits:
        - Straightforward to implement
        - Computationally efficient
        - Little memory requirements
        - Invariant to diagonal re-scaling of the gradients
        - Well suited for problems that are large in terms of data and/or parameters
        - Appropriate for non-stationary objectives
        - Appropriate for problems with very noisy and/or sparse gradients
        - Hyperparameters have intuitive interpretation and typically require little tuning

- **Adam Configuration Parameters**
    - **alpha.** Also referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training
    - **beta1.** The exponential decay rate for the first moment estimates (e.g. 0.9).
    - **beta2.** The exponential decay rate for the second-moment estimates (e.g. 0.999). This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).
    - **epsilon**. Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8).

Model Combine - Loss Function

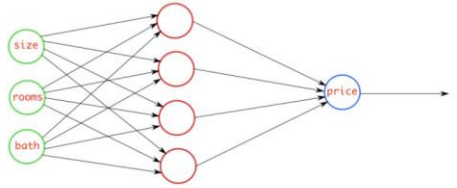


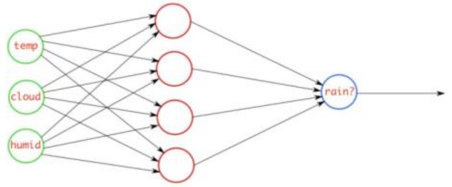
```
nn.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

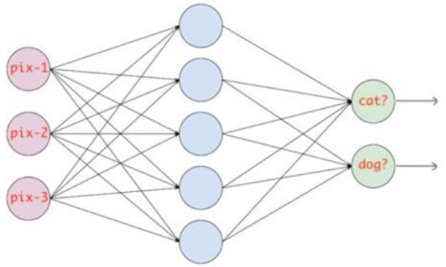
It is the prediction error of Neural Net. The loss is sued to calculate the gradients and gradients are used to update the weights of the Neural Net which is the way a neural net is trained

- Essential Loss Functions
  - MSE (Mean Squared Error)
  - Binary Crossentropy (BCE)
  - Categorical Crossentropy (CC)
  - Sparse Categorical Crossentropy (SCC)

- The loss function is calculated by taking the mean of squared differences between actual(target) and predicted values. The output is a real number
- The loss function just need one output note to classify the data into two classes. The output value should be passed through a **sigmoid activation function and the range of output is (0 or 1).**
- If using this loss function, there should be the same number of output notes as the classes. **And the final layer output should be passed through a *softmax* activation** so that each node output a probability value between (0 or 1). Use when one sample have multiple classes or labels. Output is soft probabilities.
- If you are using this function, you don't need to one hot encode the target vector. Use when you have two or more label classes.







Time to Code

Multiclass Prediction in Deep Learning Models

Suggested Time:

20 minutes

Questions?

Recap

Next Class

You will learn how to save your models for practical deployment as well as for future iterative improvements.

The End