Introduction to Advanced Machine Learning

Data Boot Camp

Lesson 21.1

Class Objectives

By the end of this lesson, you will be able to:

01

Compare traditional ML classification and regression models and the neural network models.

02

Describe the perceptron model and its components.

03

Implement neural network models using TensorFlow.

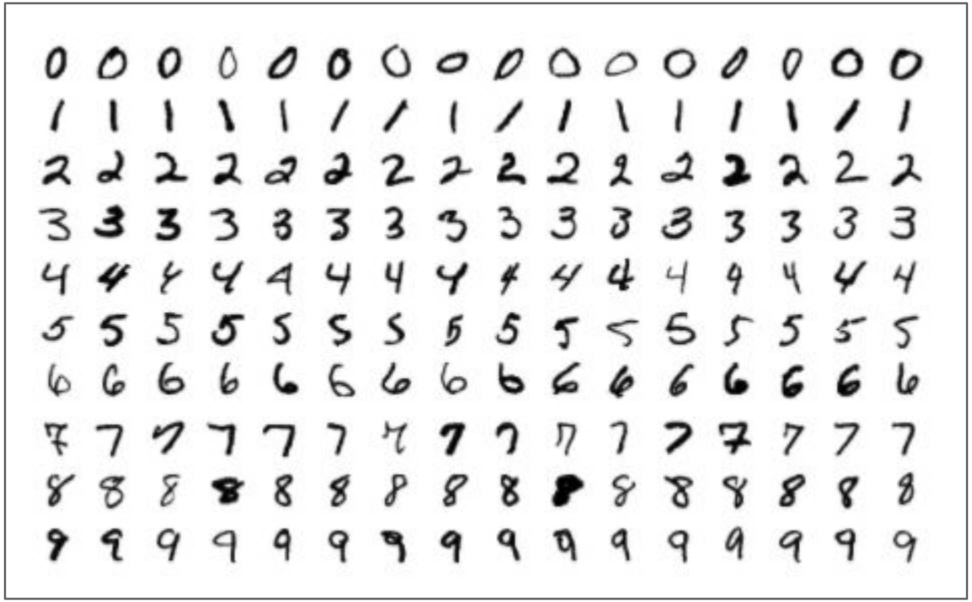
Instructor Demonstration

Surfing the Neural Net

Surfing the Neural Net

The **MNIST** database:

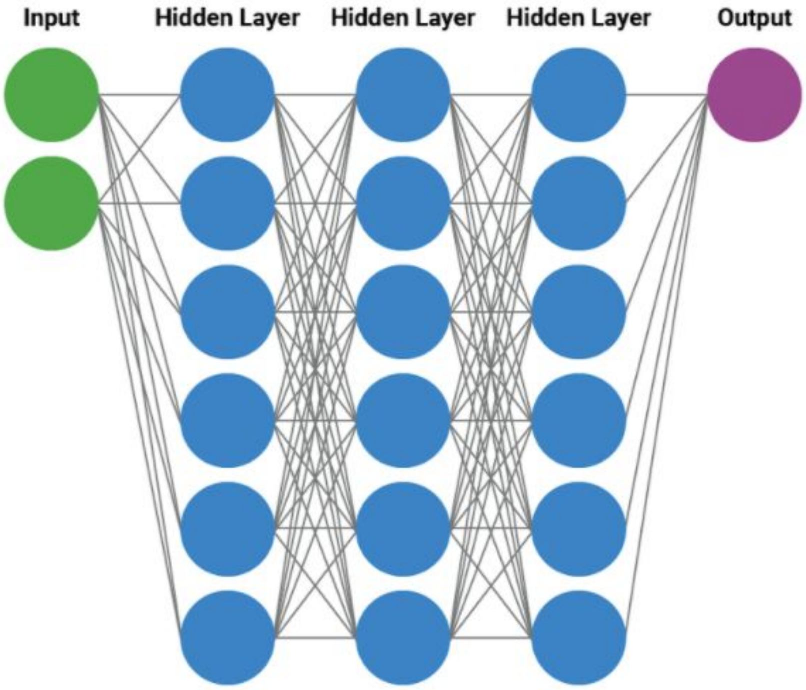
- The MNIST (Modified National Institute of Standards and Technology) dataset contains black and white images of handwritten numbers.
- A neural network can train on each pixel of each image as a scaled value from zero (completely white) to one (completely black). With enough data points, a trained neural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



Sample images from MNIST test dataset

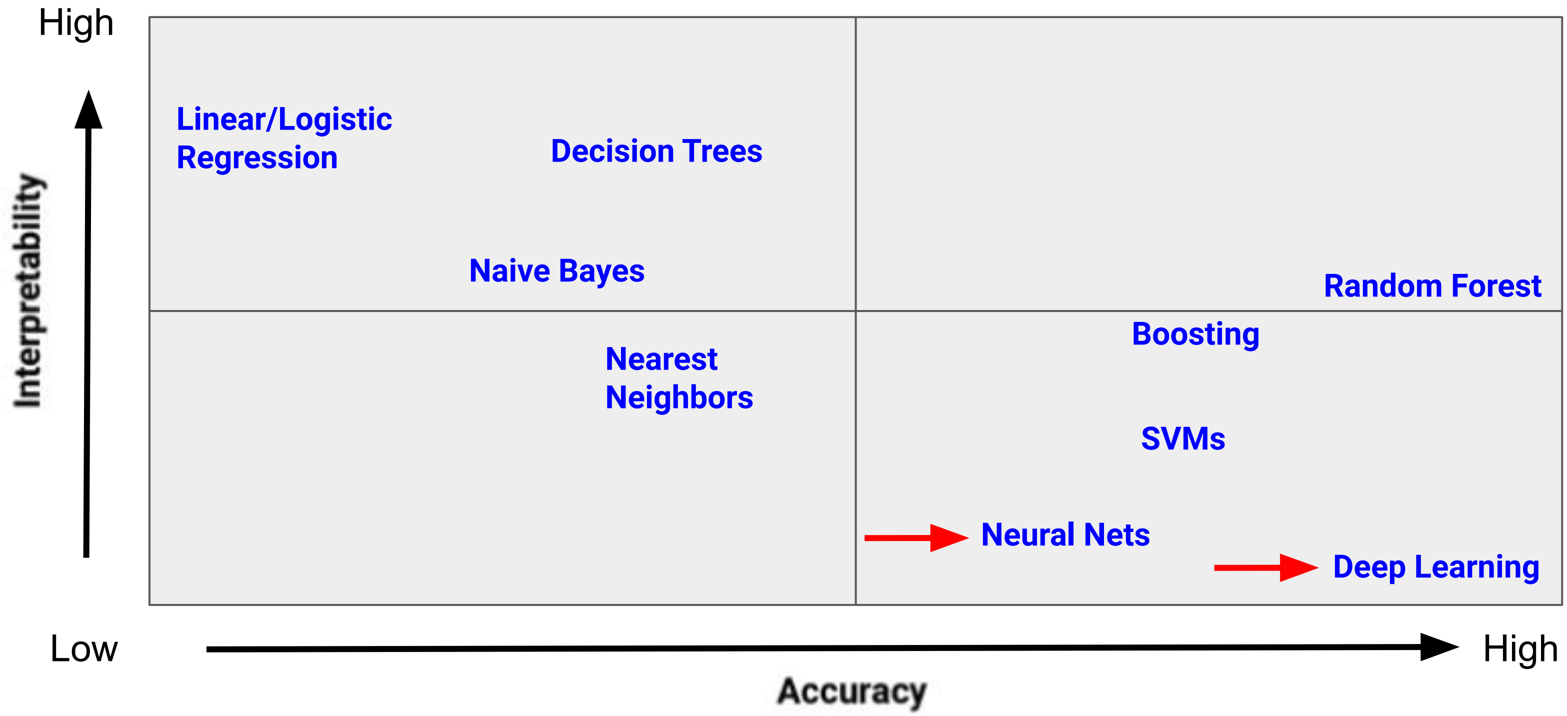
Surfing the Neural Net

Example diagram of a neural network:



Surfing the Neural Net

Different types of Machine Learning



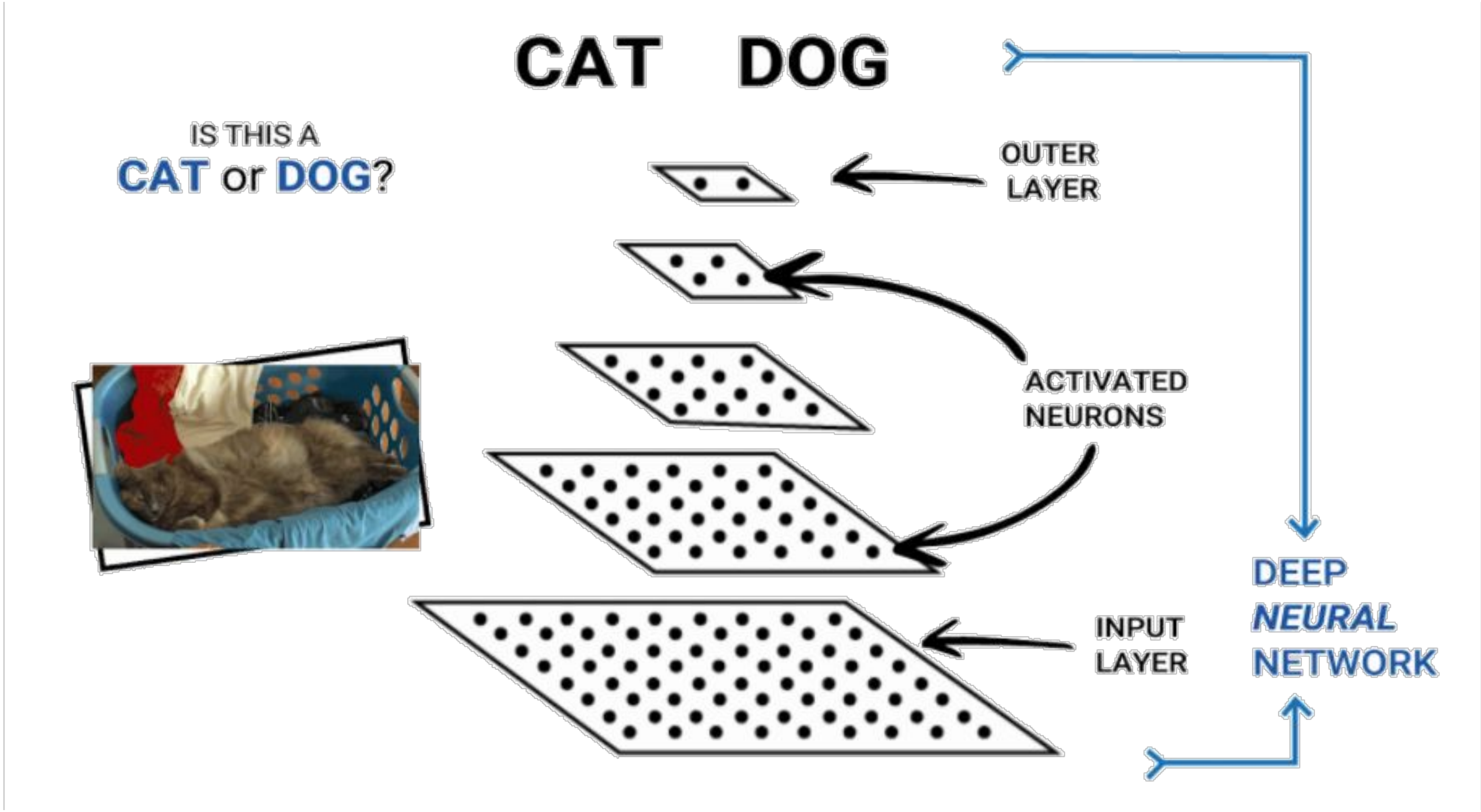
Instructor Demonstration

What is a Neural Network?

Anyone has heard of Neural Network?

Neural Network is an advanced form of machine learning that contains multiple layers of **nodes**, which perform individual computations.

What is a Neural Network?



Activity: Working through the Logistics

In this activity, you will use the logistics regression to build a binary classification model, the precursor to neural networks.

Suggested Time:

15 minutes

Instructions:

Activity: Working through the Logistics

- Use the starter code provided to create your `make_blobs` dataset from Scikit_learn.
- Split your dataset into training and testing sets using Scikit-learn's `train_test_split` module.
- Create a `LogisticRegression` instance from Scikit-learn's `LogisticRegression` model.
    - **Hint:** If you need a reminder on how to create a `LogisticRegression` model, look at Scikit-learn documentation.
- Train your `LogisticRegression` model on the training dataset.
- Evaluate your trained `LogisticRegression` model using the `accuracy_score` metric from Scikit-learn.

Time's Up! Let's Review.

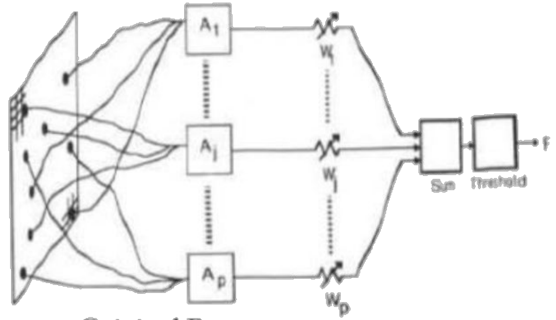
Instructor Demonstration

Perceptron, the Computational Neuron

Perceptron, the Computational Neuron

Perceptron Model

- Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron.
- A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.

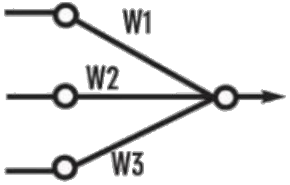


Original Perceptron

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

Frank Rosenblatt
(1928-1971)

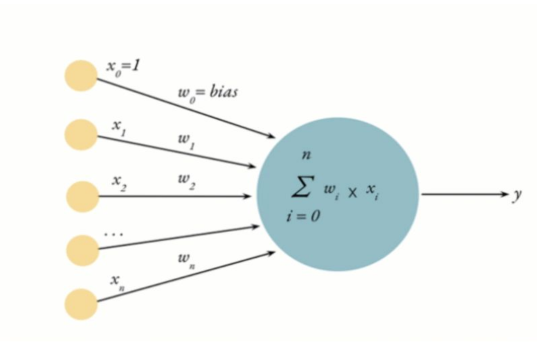
Simplified model:

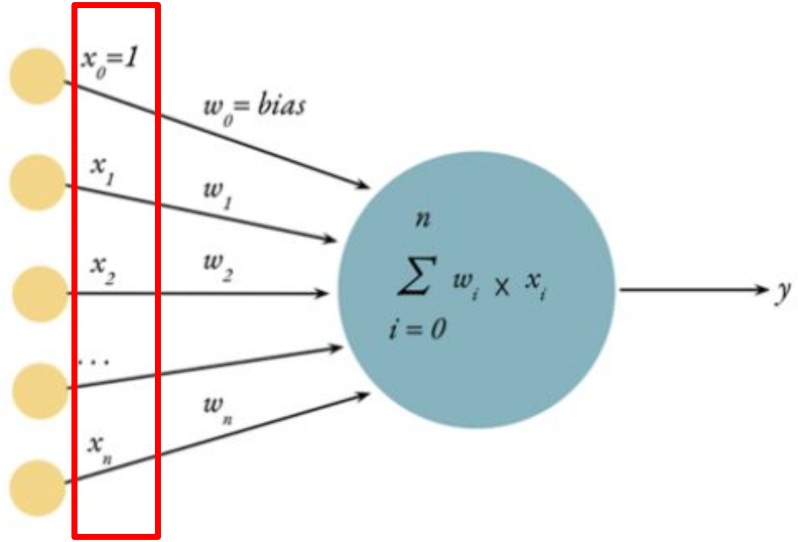


Perceptron, the Computational Neuron

The four major components of the Perceptron Model

- 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.

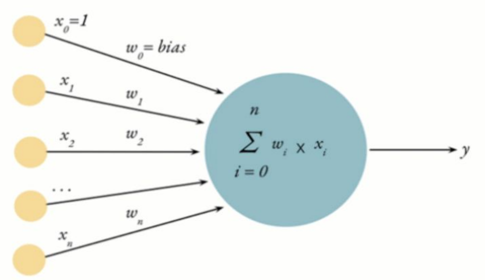


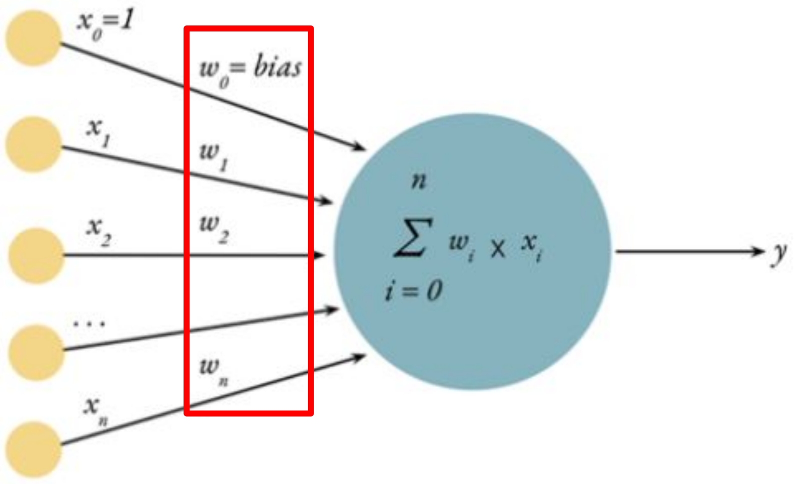


Perceptron, the Computational Neuron

The four major components of the Perceptron Model

- 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.

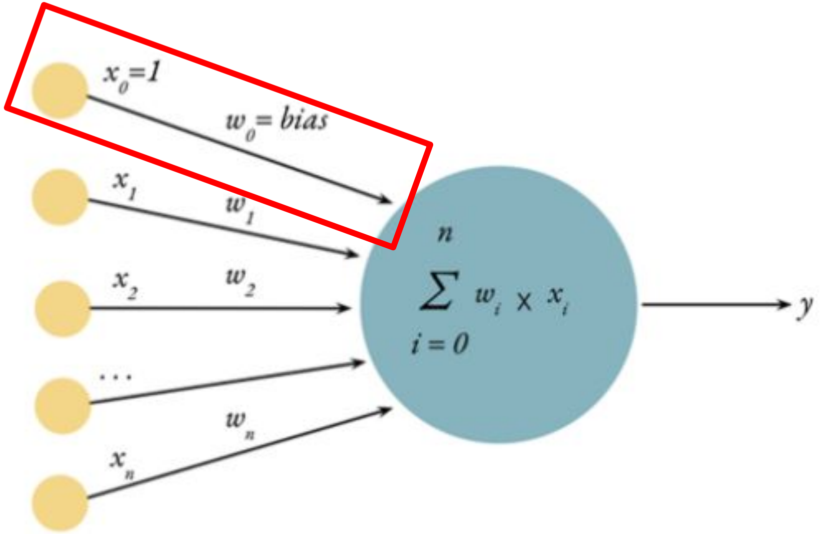


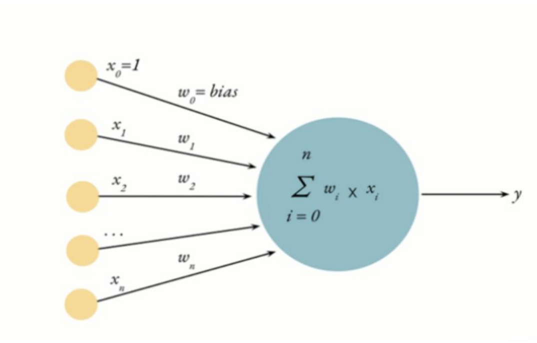


Perceptron, the Computational Neuron

The four major components of the Perceptron Model

- 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.

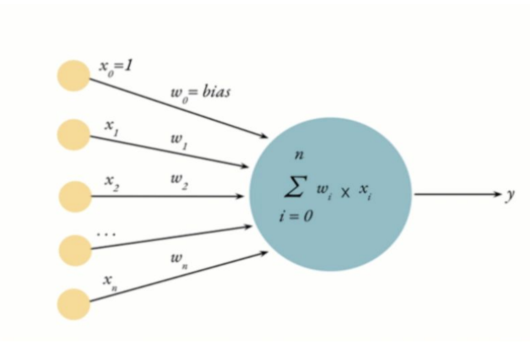


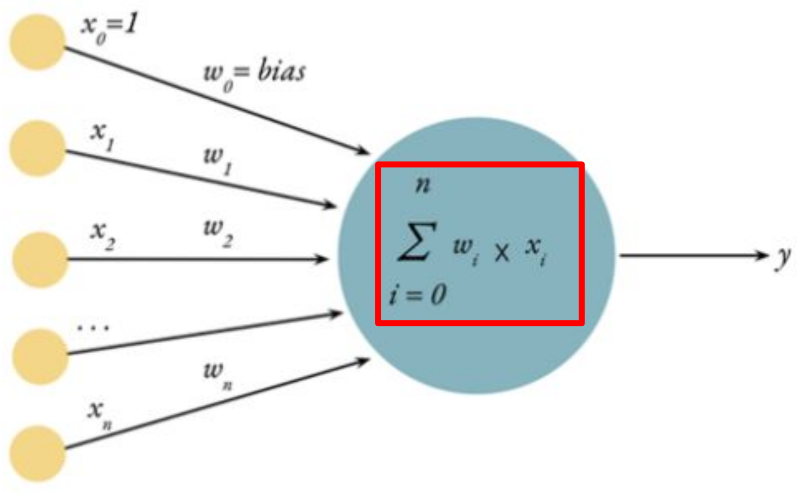


Perceptron, the Computational Neuron

The four major components of the Perceptron Model

- The **net summary function** aggregates all weighted inputs to provide an output value. In this example, the net summary function is a summation.



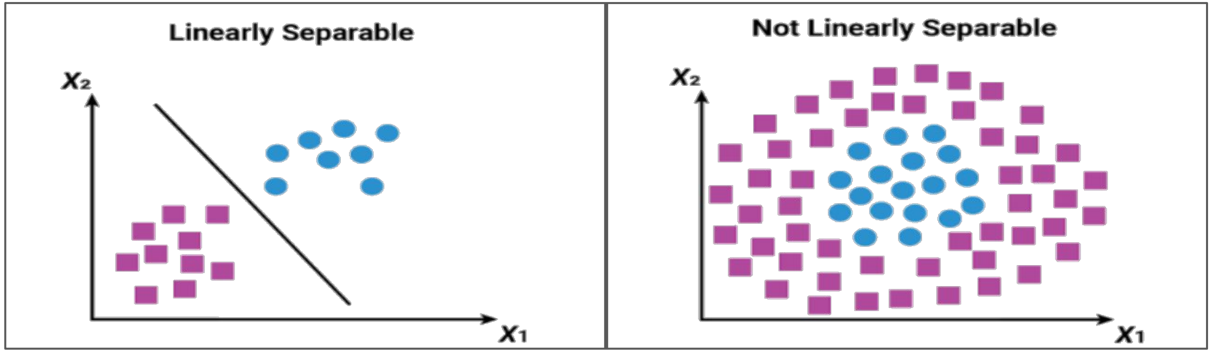


Perceptron, the Computational Neuron

Linear binary classifier - Perceptron

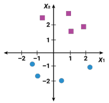
- Also known as **linear binary classifier**, is most commonly used to separate data into two groups.
- Consider **linearly separable**, the perceptron algorithm separate and classify the data into two groups using linear equation.
- Since we provide the model our input feature and parameters the perceptron model is a form of **supervised machine learning**.

➔ Comparing non linearly separable and linearly separable datasets.

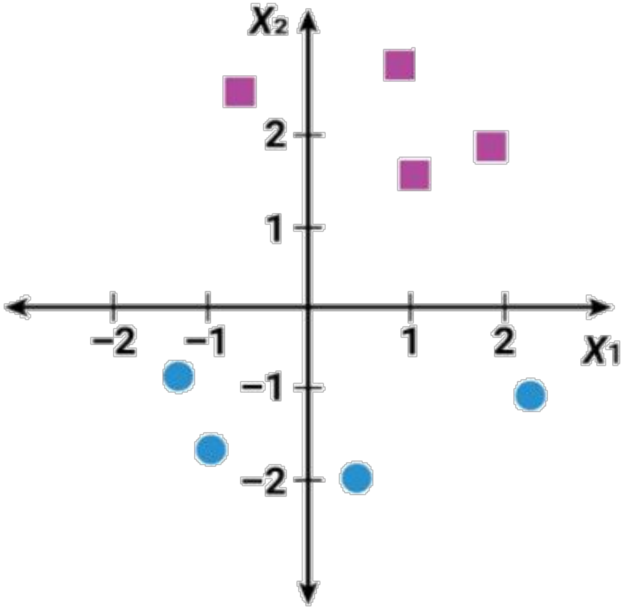


Perceptron, the Computational Neuron

Perceptron classification model

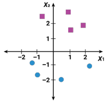


- The perceptron classification model that will distinguish between purple squares and blue circles.
- Our perceptron model will need three inputs as we are trying to classify values in a two-dimensional space.
- The inputs are:
  - ➔ ***$\chi 1$*** - the x value
  - ➔ ***$\chi 2$*** - the y value
  - ➔ ***$\omega 0$*** - the bias constant
- Net sum function:
  - ➔ **$\omega 0 + \chi 1 \omega 1 + \chi 2 \omega 2$.**

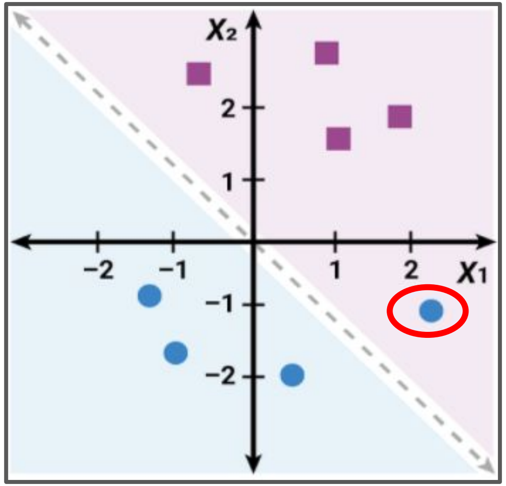


Perceptron, the Computational Neuron

Perceptron classification model



- Trained and untrained model.

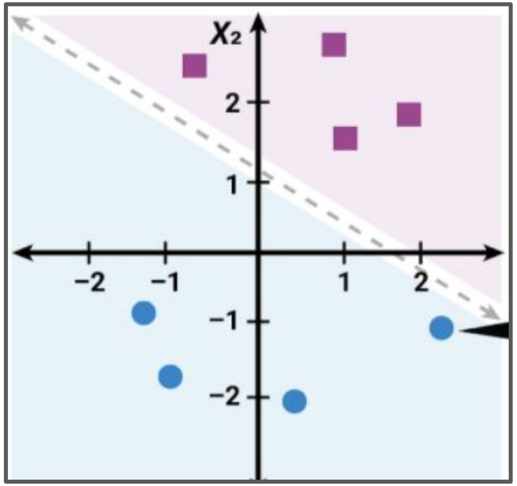


- **Untrained model**

Untrained model almost classified the two groups perfectly. It misclassified one blue dot.

Evaluation of each data point in order 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.

- **Trained model**



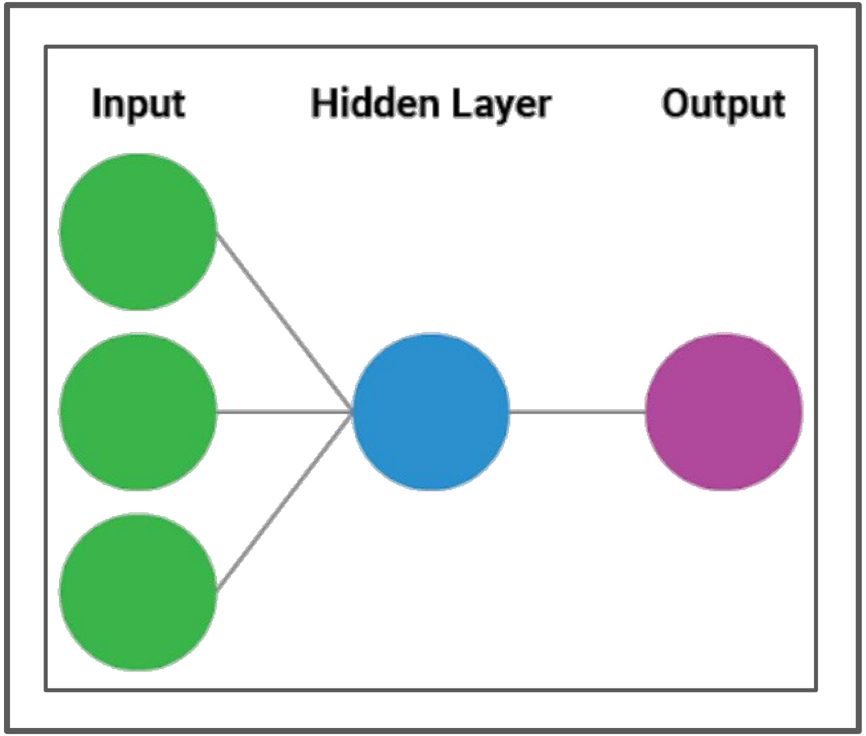
Instructor Demonstration

Make the Connections in a Neural Network

Make the Connections in a Neural Network

The Structure of a Neural Network

- A basic Neural Network is composed by three layers:
  - **Input layer:** inputted values derived from transformed weight coefficients.
  - **Hidden layer:** single hidden layer of either single or multiple neurons.
  - **Output layer:** reports the classification or regression value.



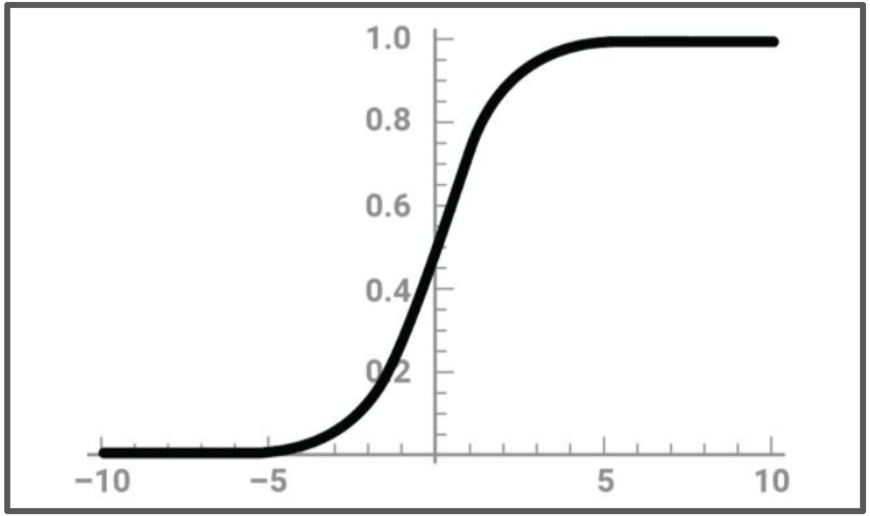
If each neuron has its own output, how does the neural network combine each neuron's output into the model's classification or regression output?

Neural Networks use an **activation function** to transform the output of each neuron to a quantitative value.The transform output is used as an input value for the other layers in the neural network model.

Make the Connections in a Neural Network

Activation Functions

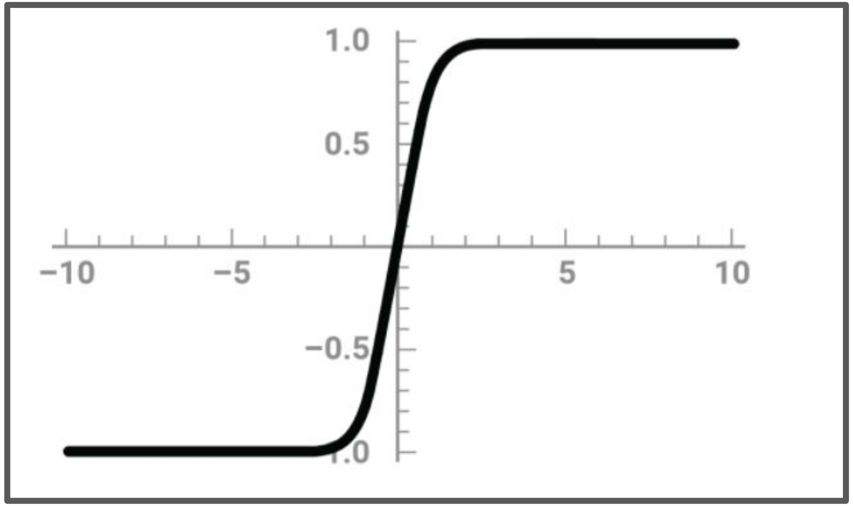
- Despite a vast range of activation functions available for many specific purposes, most neural networks will use one of the following activation functions:
  - The **linear function** transforms the output into the coefficients of a linear model (the equation of a line).
  - The **sigmoid function** is identified by a characteristic S curve. It transforms the output to a range between 0 and 1.



Make the Connections in a Neural Network

Activation Functions

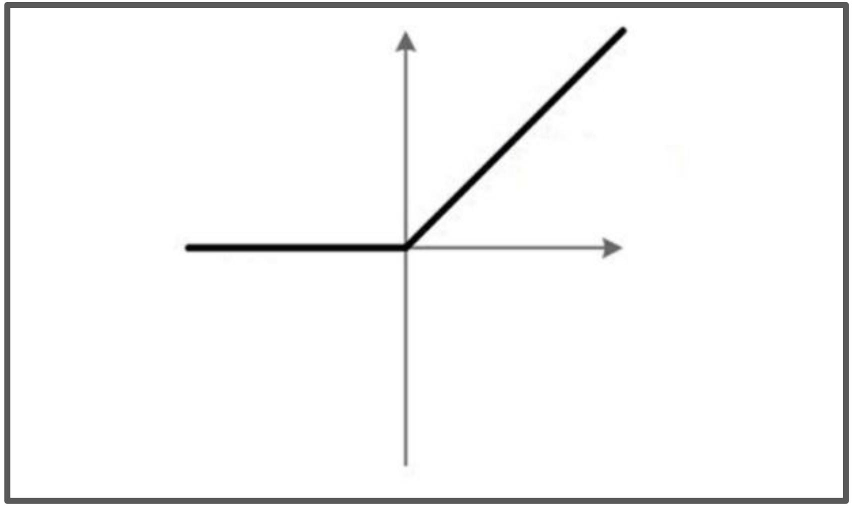
- The **tanh function** is also identified by a characteristic S curve; however, it transforms the output to a range between -1 and 1.



Make the Connections in a Neural Network

Activation Functions

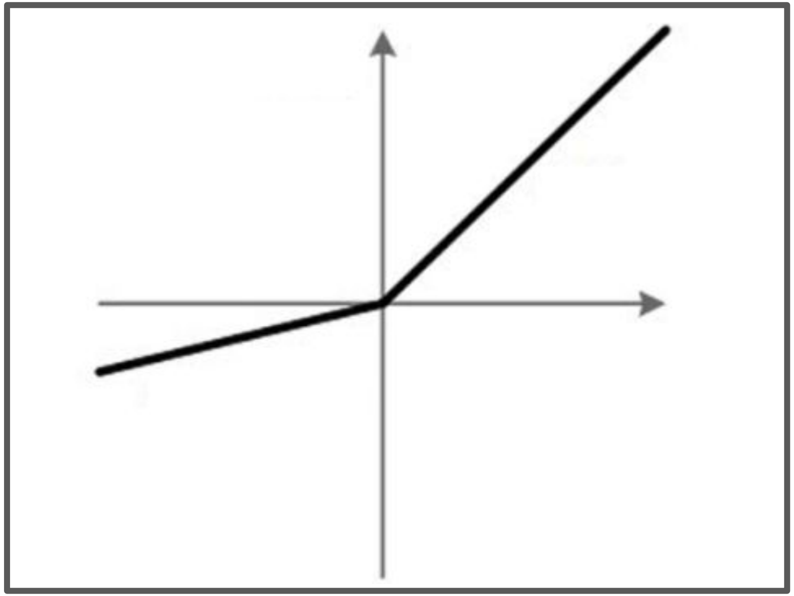
- The **Rectified Linear Unit (ReLU)** function returns a value from 0 to infinity, so any negative input through the activation function is 0. It is the most used activation function in neural networks due to its computational simplicity and effectiveness, but it might not be appropriate for simpler models.



Make the Connections in a Neural Network

Activation Functions

- The **Leaky ReLU** function is a "leaky" alternative to the ReLU function, whereby the negative input values will return very small, non-zero, negative values.



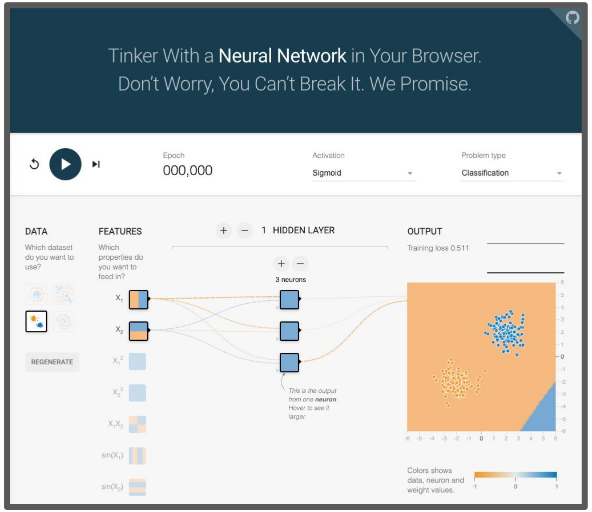
Everyone Do:

Playing in Tensorflow Playground

Playing in Tensorflow Playground

Overview

- Explore all of the different components of a neural network and how each component interact with others utilizing **Tensorflow Playground**.
- Tensorflow is an end-to-end Python library for machine learning that has become an industry standard for developing robust neural network models.

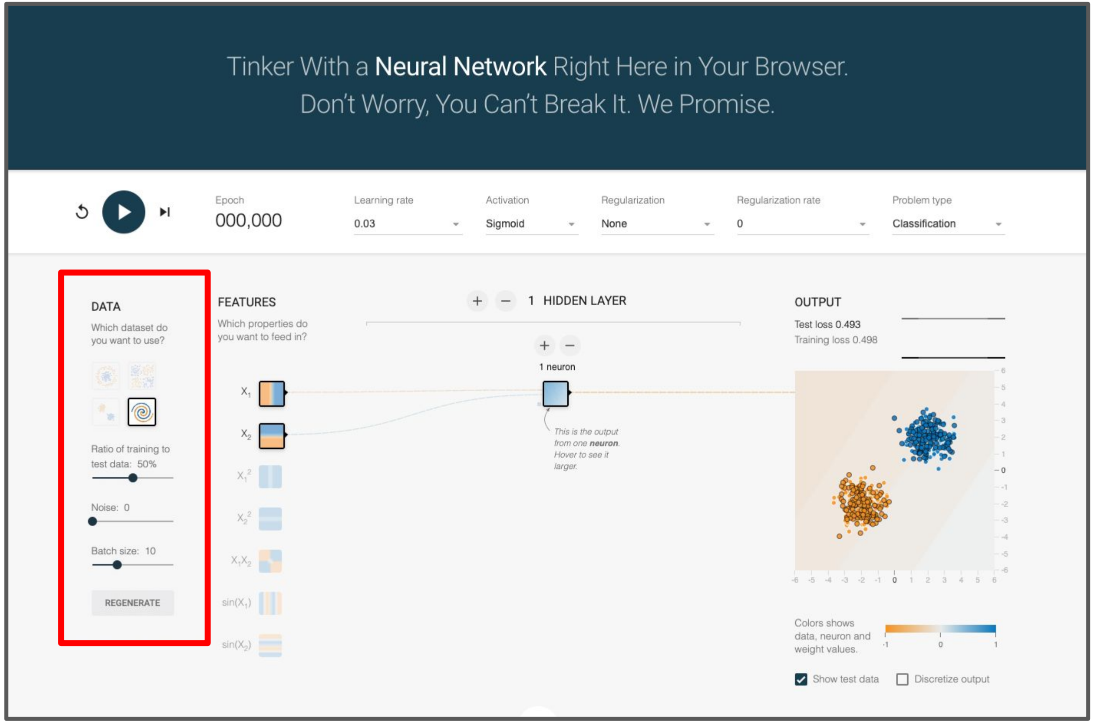


- **Tensorflow Playground** is an application developed by TensorFlow as a teaching tool to "demystify the black box" of neural networks.
  - It provides a working simulation of a neural network as it trains on a variety of different datasets and conditions.
- As an added bonus, we can also use Tensorflow Playground to test different configurations of our neural network models as an abstract form of our model - fit - predict workflow.

MODEL > FIT > PREDICT WORKFLOW

Playing in Tensorflow Playground

Playground components



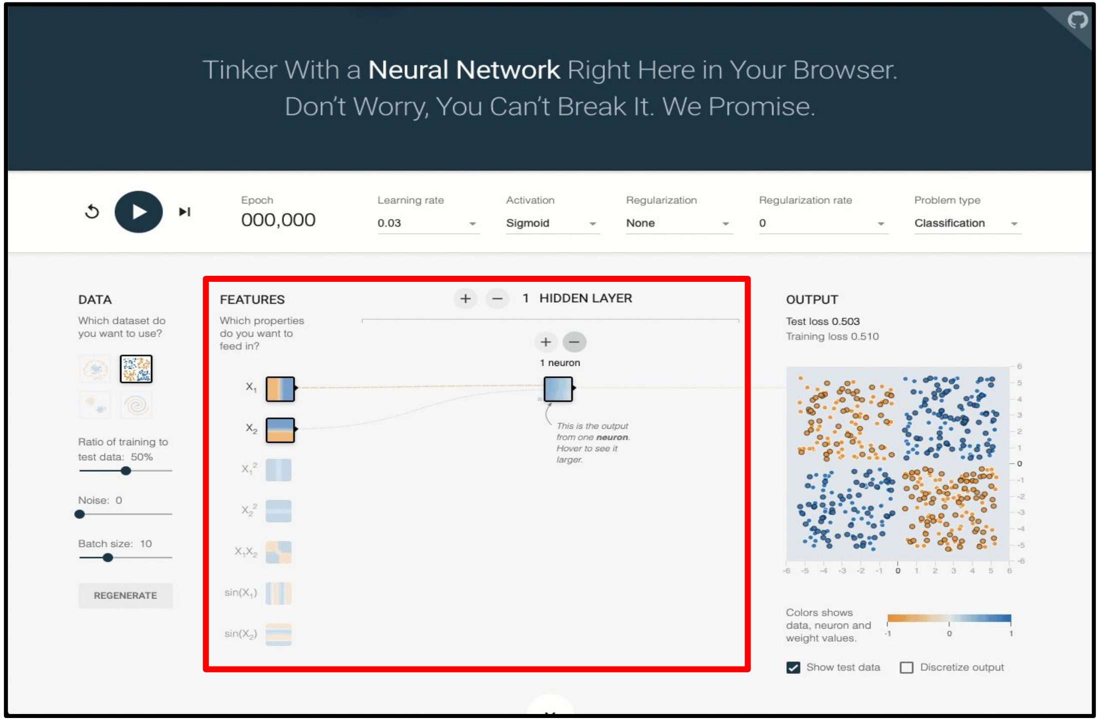
- **Input data**

Mainly, Six datasets are provided.

➔ **Classification**: Circle, Exclusive or, Gaussian, Spiral.
➔ **Regression**: Plane, Multi Gaussian.

Playing in Tensorflow Playground

Playground components



- **Input features and layers**

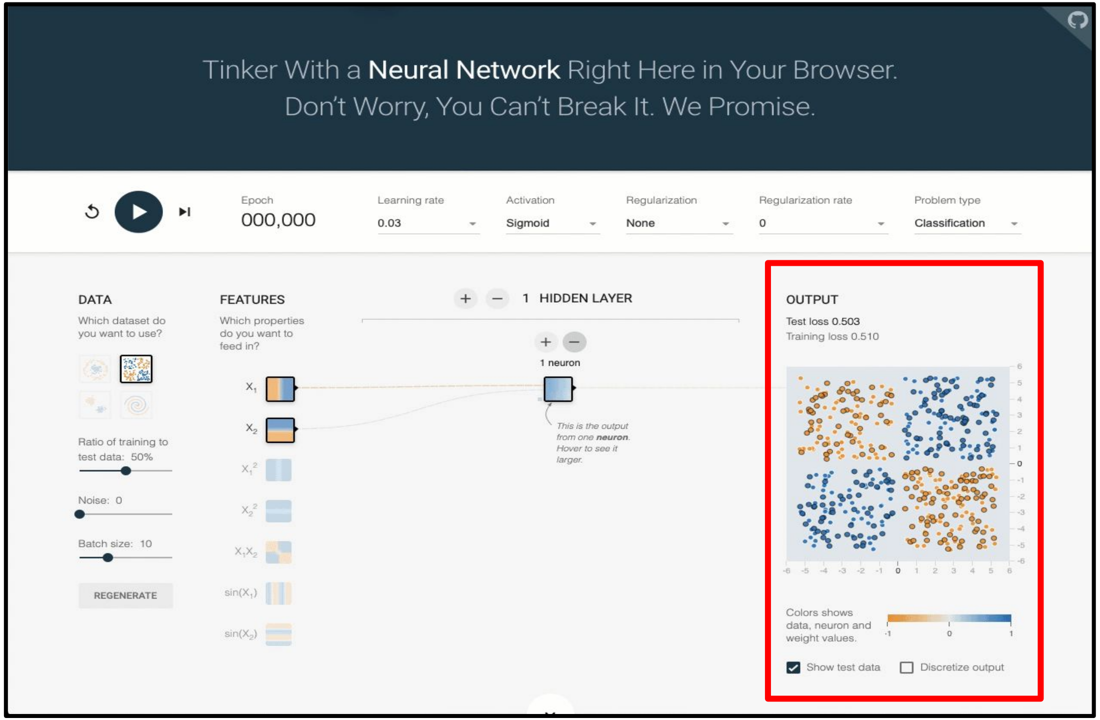
Provides seven features:

- ➔ X_1 X_2
- ➔ Squares of X_1 X_2
- ➔ Product of X_1X_2
- ➔ Sin of X_1 X_2

Neurons can be added and subtracted using the plus and the minus sign.

Playing in Tensorflow Playground

Playground components



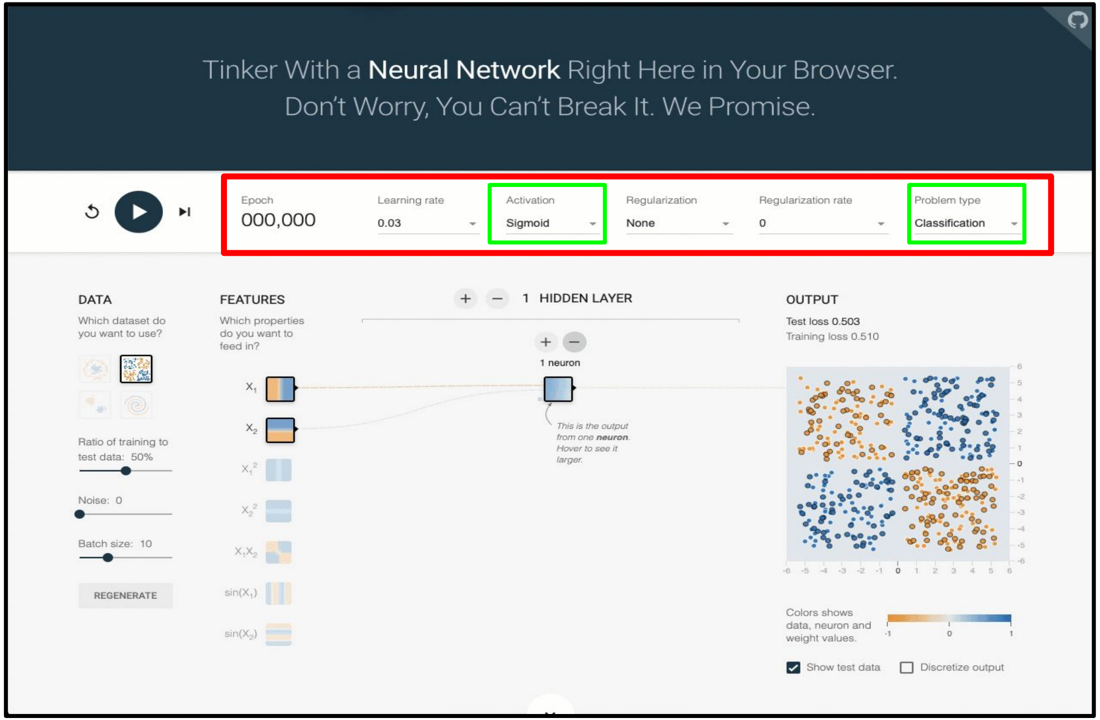
- **Output**

➔ Here you can check the model performance after training the neural network.

➔ We want to take a closer look at the Test loss function: the better the model performs, the lower the test loss value is.

Playing in Tensorflow Playground

Playground components



- **Simulation parameters**

→ At the top of page we find the simulation parameters.

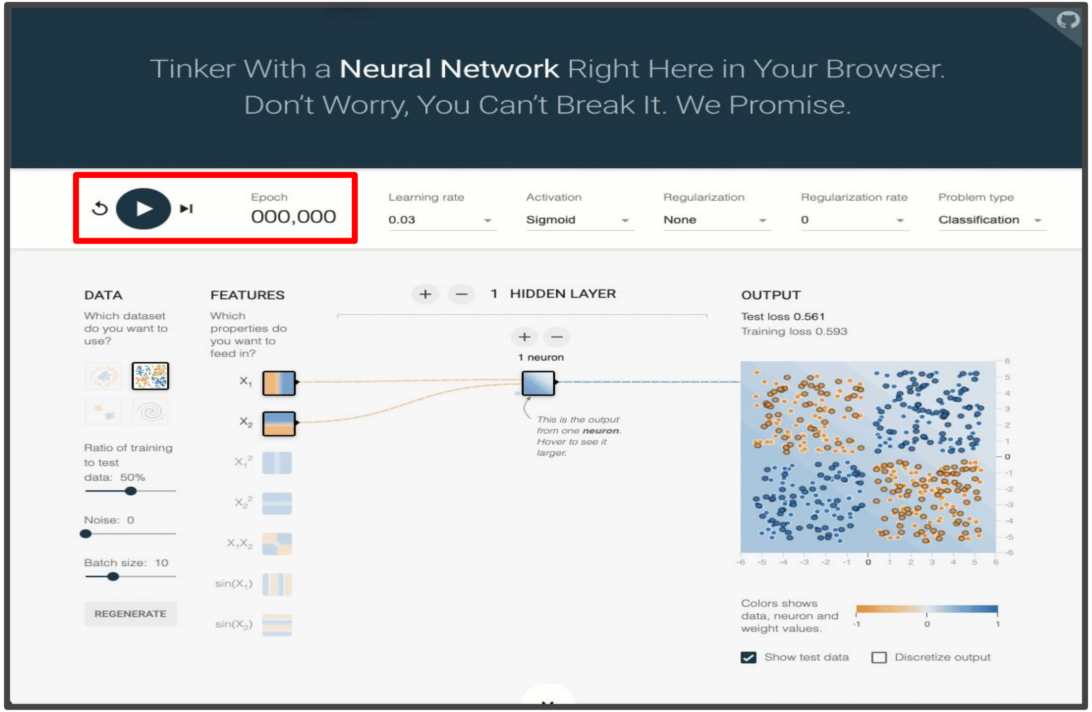
→ We can find five different 'classes' of parameters where we are able to set to perform our test. The classes are:

- **Learning rate**
- **Activation**
- **Regularization**
- **Regularization rate**
- **Problem type**

→ Note that for our purposes we are only going to concentrate on the **activation** and **problem** type parameters.

Playing in Tensorflow Playground

Playground components



- **Simulation controls**

➔ On the left side of the simulation parameters we can find the the simulation controls and the epoch counter.

➔ Each epoch is a single training iteration in TensorFlow machine learning training.

➔ We can start simulating epochs by the model by pressing the start button until we press the pause button.

Instructor Demonstration

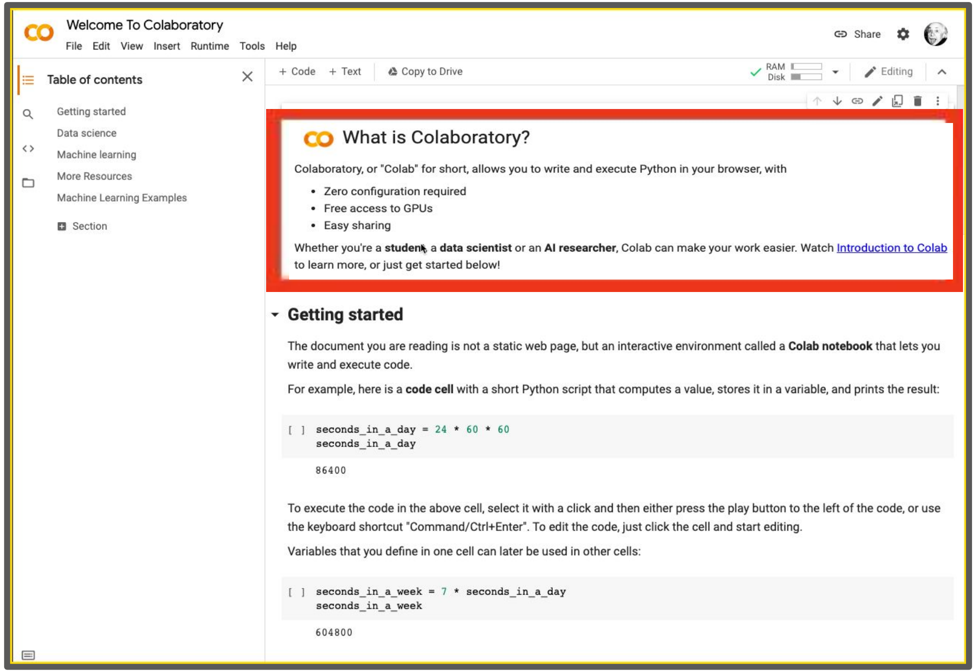
Set up Google Colab

Set up Google Colab

What is and why use Google Colaboratory?

- 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

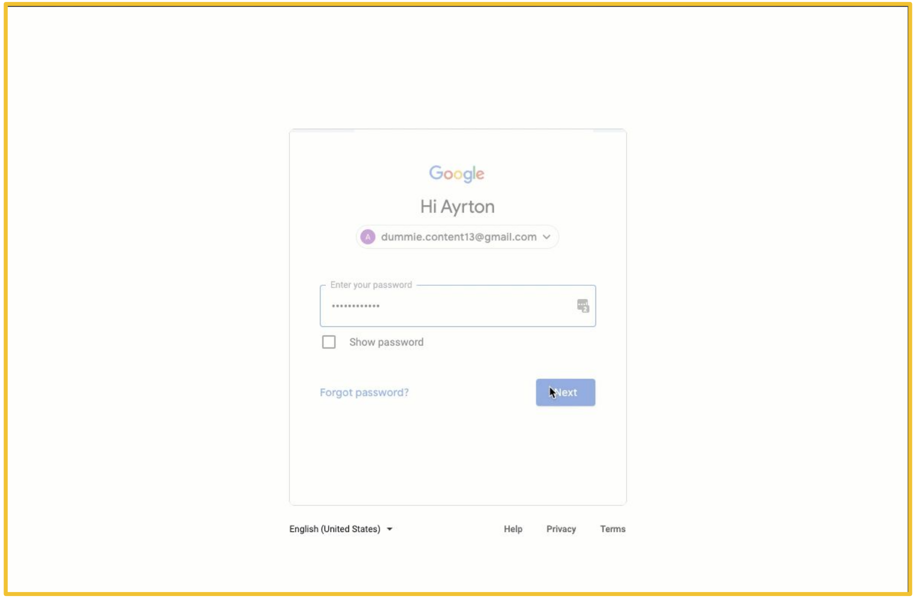
- Google Colab is a cloud-based notebook, different from Jupyter Notebook, which runs locally in our machine.
- Cloud-based notebooks are user friendly as won't require any type of module installation.
- TensorFlow was also developed by Google, which makes running in Colab fairly seamless.



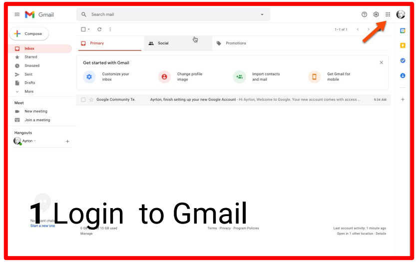
Set up Google Colab

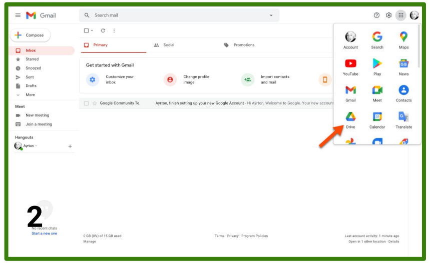
Starting with Google Colab!

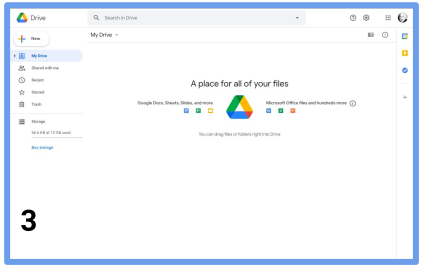
- For this step we need to have a Google account. In case you don't have one, sign up for a Google account before moving forward.



1. Navigating to Google Drive








Break
Countdown timer
15:00
(with alarm)

Everyone Do:

Work Through A Neural Network Workflow

Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

➔ Importing our dependencies.

```
%matplotlib
# Import our dependencies
import pandas as pd
import matplotlib as plt
from sklearn.datasets import make_blobs
import sklearn as skl
import tensorflow as tf
```

Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

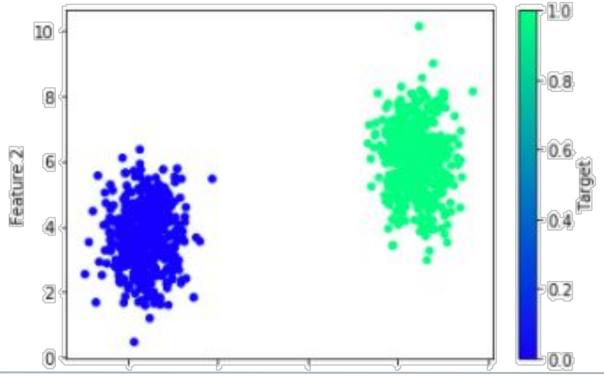
➔ Generating and visualizing our dummy data.

- Second, using Scikit-learn's `make_blobs` method we are going to create our dummy data.
- Use `make_blobs` to create 1000 samples with two features that are linearly separable.
- Our two feature dataset will be know as `x` and `y` values.

```
# Generate dummy dataset
X, y = make_blobs(n_samples=1000, centers=2, n_features=2, random_state=78)

# Creating a DataFrame with the dummy data
df = pd.DataFrame(X, columns=["Feature 1", "Feature 2"])
df["Target"] = y

# Plotting the dummy data
df.plot.scatter(x="Feature 1", y="Feature 2", c="Target", colormap="winter")
```



Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

➔ Splitting the dummy data into training and test datasets.

- Next. separate our dataset into training and test datasets using `train_test_split` method.

```
# Use sklearn to split dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=78)
```

➔ Normalizing the dummy data.

- Once we have our training data, it’s crucial to normalize our numerical variables in order to our neural networks to not focus on outliers applying proper weights to each input.

```
# Create scaler instance
X_scaler = skl.preprocessing.StandardScaler()

# Fit the scaler
X_scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

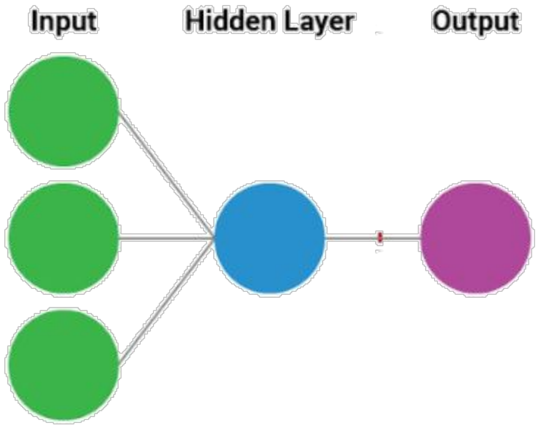
Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

➔ Creating our Sequential model.

- We are now ready to build the neural network in Python, since our dataset is preprocessed.
  - The `nn_model` object will store the entire architecture of our neural network model.

```
# Create the Keras Sequential model
nn_model = tf.keras.models.Sequential()
```



Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

➔ Creating the first Dense layer.

- The process of building our layers by combining the input layer with the first hidden layer is simplified by using the Keras module concentrating in these 3 parameters:
  - `units`: indicates how many neurons we want in the hidden layer.
  - `activation`: indicates which activation function to use.
  - `input_dim`: indicates how many inputs will be in the model.

```
# Add our first Dense layer, including the input layer
nn_model.add(tf.keras.layers.Dense(units=1, activation="relu", input_dim=2))
```

Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

➔ Adding the output Dense layer.

- Since we are trying to build a neural network classification model, we want the activation function of the output layer to be the **sigmoid** activation function to produce a probability output.

```
# Add the output layer that uses a probability activation function
nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))
```

➔ Checking the structure of our model.

- Double-check our model structure using the `summary` method.

```
# Check the structure of the Sequential model
nn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 3
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2
=================================================================
Total params: 5
Trainable params: 5
Non-trainable params: 0
_________________________________________________________________
```

Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

➔ Compiling and training the neural network model.

- **Note:**
  - Depending on the function of the neural network, we will need to compile and train the neural network model with a specific l**oss metric, optimization function,** and **evaluation metric**.
  - **TensorFlow** and **Keras** have many parameters to tweak the performance, but most basic classification and regression models use the same parameters.
  - For all of our models, we will always use the `adam` optimizer. The `metrics` parameter is used to print a performance metric at the end of each epoch, so that we can judge how well the model is doing during training.

```
# Compile the Sequential model together and customize metrics
nn_model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])

# Fit the model to the training data
fit_model = nn_model.fit(X_train_scaled, y_train, epochs=100)
```

Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

➔ Visualizing the model’s training loss over 100 epochs.

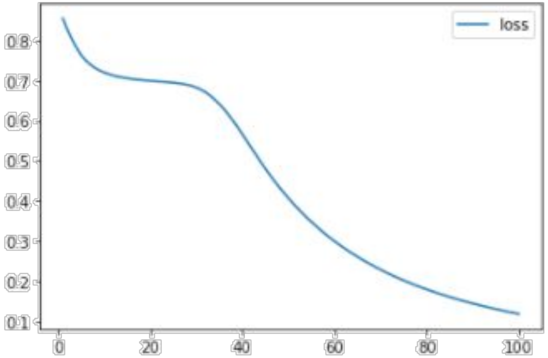
- At this point we have a trained neural network model. To check our model does not need retraining we need to test the performance of our model.

```
# Create a DataFrame containing training history
history_df = pd.DataFrame(fit_model.history)

# Increase the index by 1 to match the number of epochs
history_df.index += 1

# Plot the loss
history_df.plot(y="loss")
```

```
<AxesSubplot:>
```



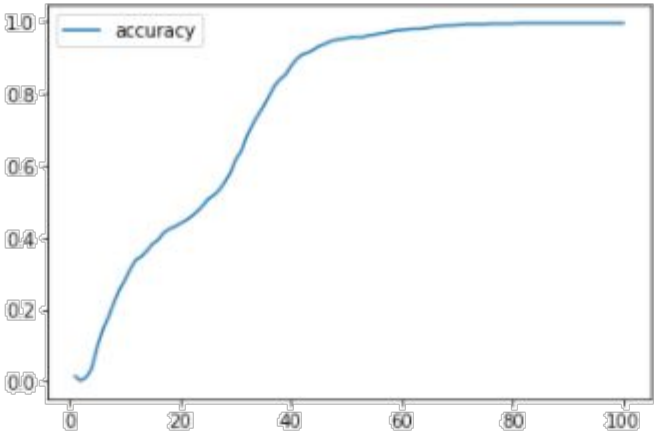
Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

➔ Visualizing the model's predictive accuracy over the same timeframe.

```
# Plot the accuracy
history_df.plot(y="accuracy")
```

```
<AxesSubplot:>
```



Work Through A Neural Network Workflow

Building our first neural network model in TensorFlow

➔ Evaluating the test loss and predictive accuracy of the model on our testing dataset.

- The final step of workflow is to evaluate the performance of the trained model against the test dataset.
- When it comes to a classification model, we want our neural network to have a predictive accuracy as close to 100% or 1.0

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
8/8 - 0s - loss: 0.0683 - accuracy: 1.0000
Loss: 0.06833964586257935, Accuracy: 1.0
```

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:

20 minutes

Instructions:

Activity: BYONNM - Build Your Own Neural Network Model

- 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.


RECAP