

Deep Learning with Keras and TensorFlow



TensorFlow



Learning Objectives

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

- 🕒 Install and setup TensorFlow and TFLearn for building and testing models
- 🕒 Gain hands-on experience with TensorFlow Playground to visualize and understand the impact of hyperparameters on model performance
- 🕒 Explore various applications of TensorFlow across different industries, such as healthcare and social media
- 🕒 Analyze the workflow of TFLearn, that includes creating input layers, configuring models, and training neural networks
- 🕒 Evaluate and improve machine learning models using TensorFlow and Keras



Business Scenario

XYZ Corporation is a technology company that specializes in developing advanced artificial intelligence (AI) applications. To stay ahead of the competition, it has decided to use TensorFlow, an open-source library for deep learning.

It intends to use TensorFlow's comprehensive suite of tools to train multiple neural networks, TensorBoard for the visualization of computational graphs, and Keras for the easy and efficient creation of neural network models.

With these tools, XYZ Corporation can develop sophisticated AI applications and gain a competitive edge in the fast-paced technology industry.





Introduction to TensorFlow

What Is TensorFlow?

TensorFlow is an open-source, Python-compatible toolkit for numerical computation that accelerates and improves the creation of neural networks and machine learning algorithms.

A popular open-source library
for deep learning

Developed by the Google Brain
team and released in 2015

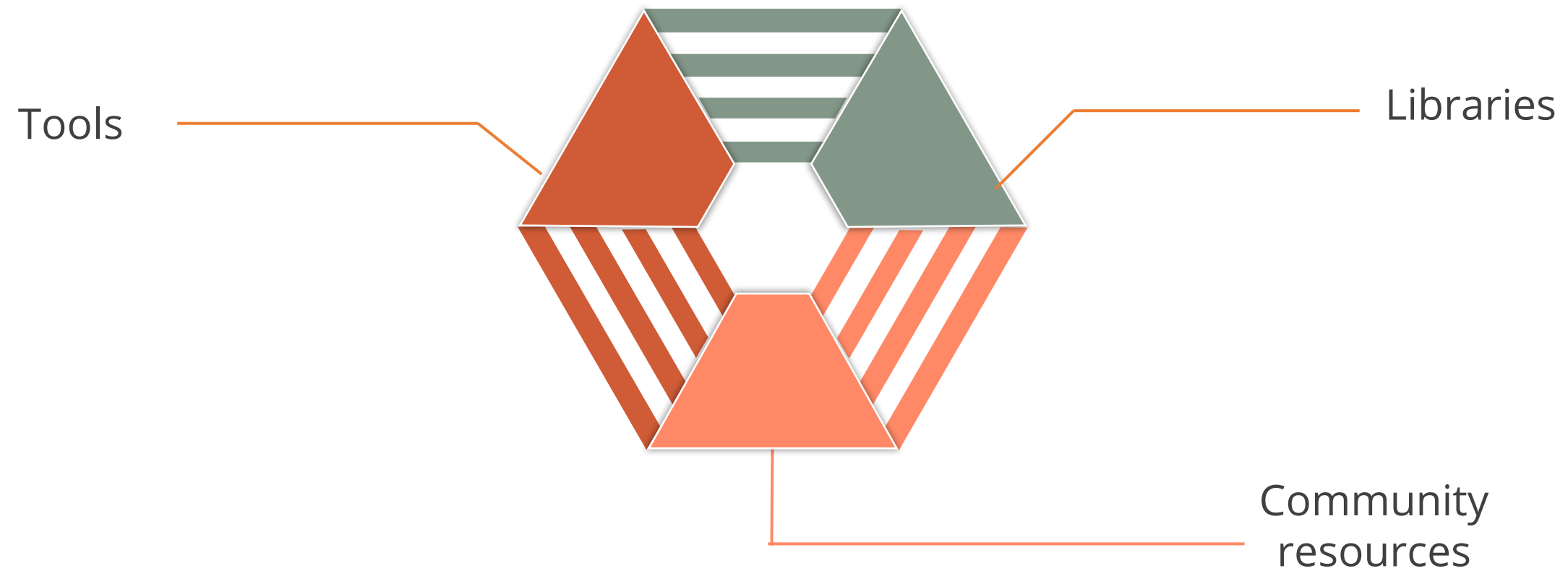


Used mainly for classification,
prediction, and creation of models

TensorFlow

It is an effective and adaptable machine learning library that enables researchers to build innovative ML models and quickly deploy applications.

It has the following:



TensorFlow

In TensorFlow, the data values are not stored as integers, floats, or strings but are encapsulated in an object called a tensor.



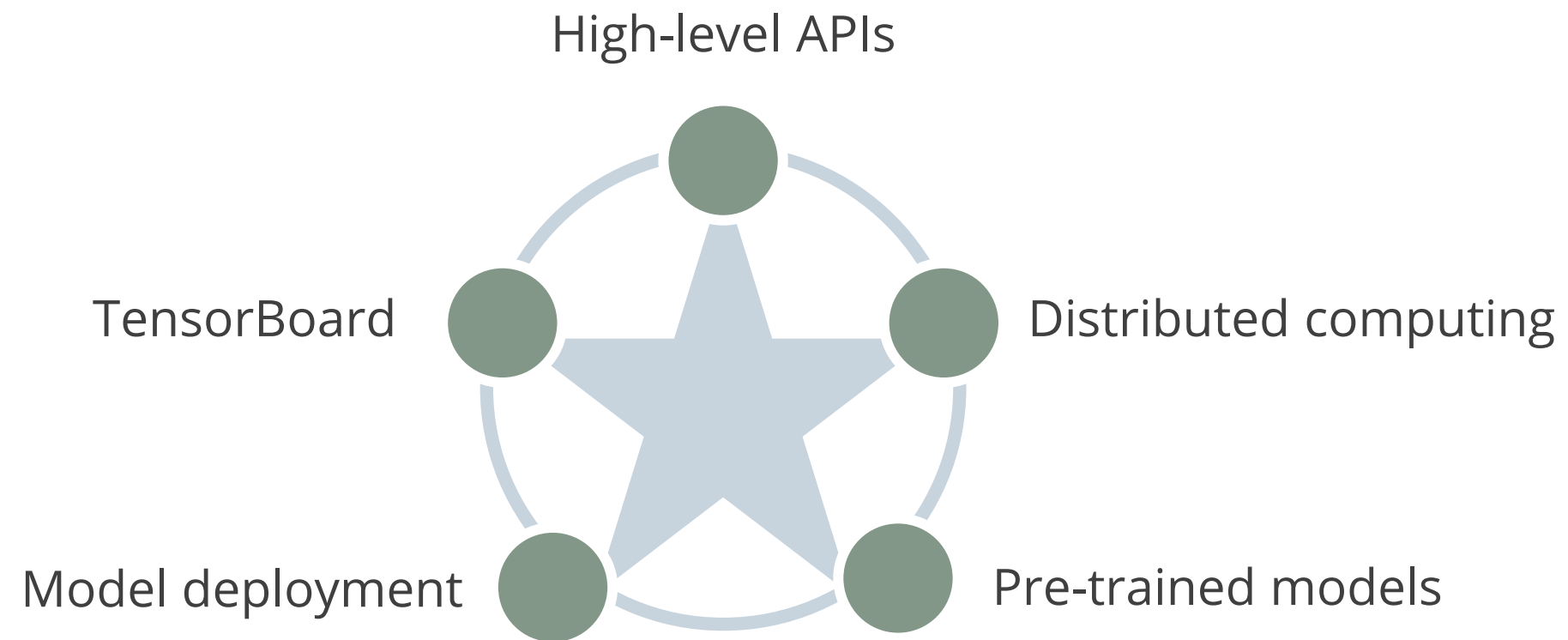
Example of a tensor

If a Python list is passed to TensorFlow, it will be converted into a tensor.

TensorFlow

It is a powerful AI tool that enables the creation of large-scale neural networks with complex layers.

Features of TensorFlow:



Why Is TensorFlow Necessary?

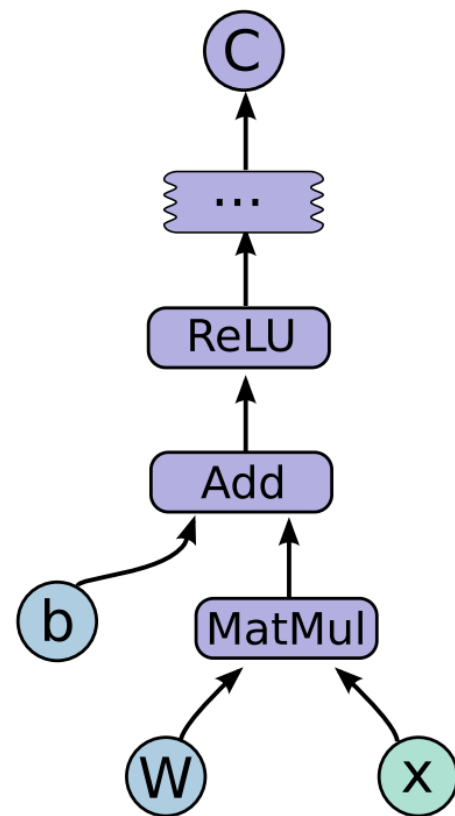
TensorFlow operates on a system of data flow graphs, which allows for efficient computation and parallel processing. It is essential for handling the heavy computational requirements of deep learning.



It provides a comprehensive and flexible framework for designing, training, and deploying deep learning models.

TensorFlow: Dataflow Graph

TensorFlow is a library for numerical computation, that uses dataflow graphs.



Nodes represent mathematical operations.

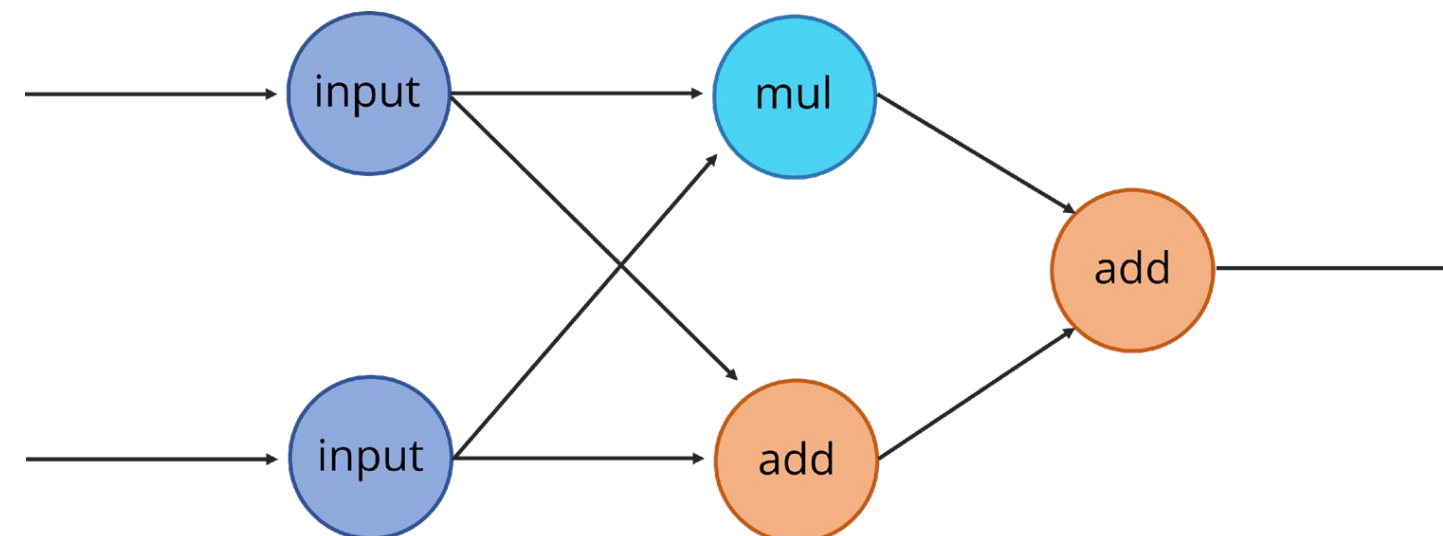
Edges represent multidimensional data arrays (tensors) transferred between nodes in the graph.

TensorFlow: Dataflow Graph

TensorFlow uses a dataflow graph to represent computation.

Dataflow is a common programming model for parallel computing.

TensorFlow optimizes graph execution by rearranging operations and leveraging parallelism for efficient computation.



Benefits of Using Dataflow Graphs

Parallelism

It is easy for the ML system to identify operations that can be executed in parallel.

Distributed execution

It is possible for TensorFlow to distribute programs across multiple devices, CPUs, and GPUs.

Compilation

It helps to quickly generate code.

Portability

It can be built in Python, stored in a saved model, and restored in a C++ program.

Categories of TensorFlow APIs

TensorFlow APIs can be divided into two broad groups:

Low-level API

Example: TensorFlow Core

It is recommended for deep learning researchers.

It provides finer levels of control over the models.

High-level API

Example: `tf.keras`, `tf.estimator`

It makes TensorFlow easier to use, without sacrificing flexibility and performance.

It provides tools for building and training complex models with just a few lines of code.

TensorFlow: Features



It is an open-source library that enables efficient and accelerated computations.

It supports pipelining and multi-GPU training, enhancing efficiency for large-scale models.

It serves as an intermediary between raw input data and models for efficient processing.

It offers built-in visualization, mainly through TensorBoard, for easy analysis and visualization of computational graphs.

Advantages of TensorFlow



Flexibility



Parallel computation



Friendly with multiple environments



Open-source platform



High-level abstraction

TensorFlow: Flexibility



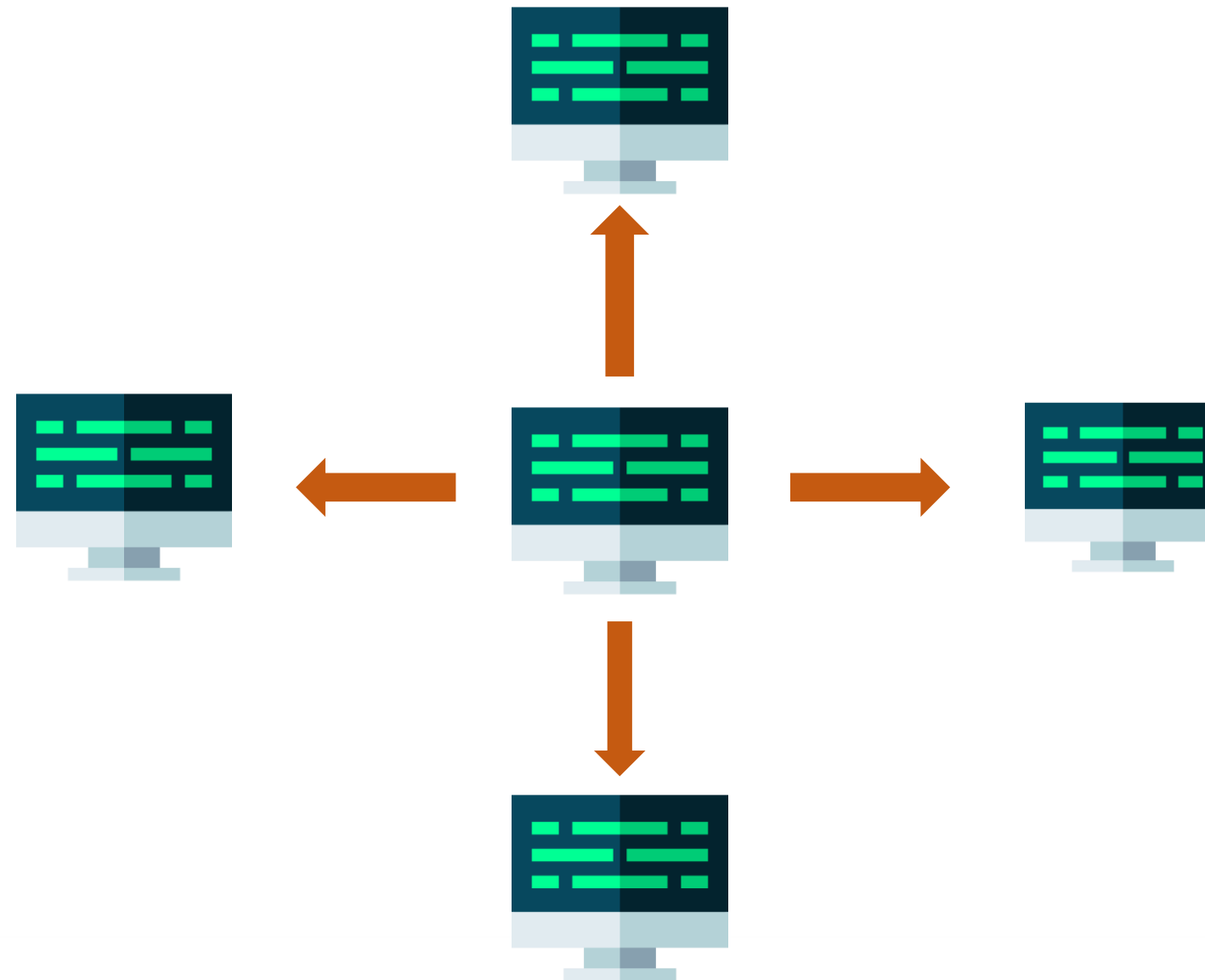
Python API offers flexibility to create all sorts of computations for every neural network architecture.



It includes highly efficient C++ implementations of many ML operations.

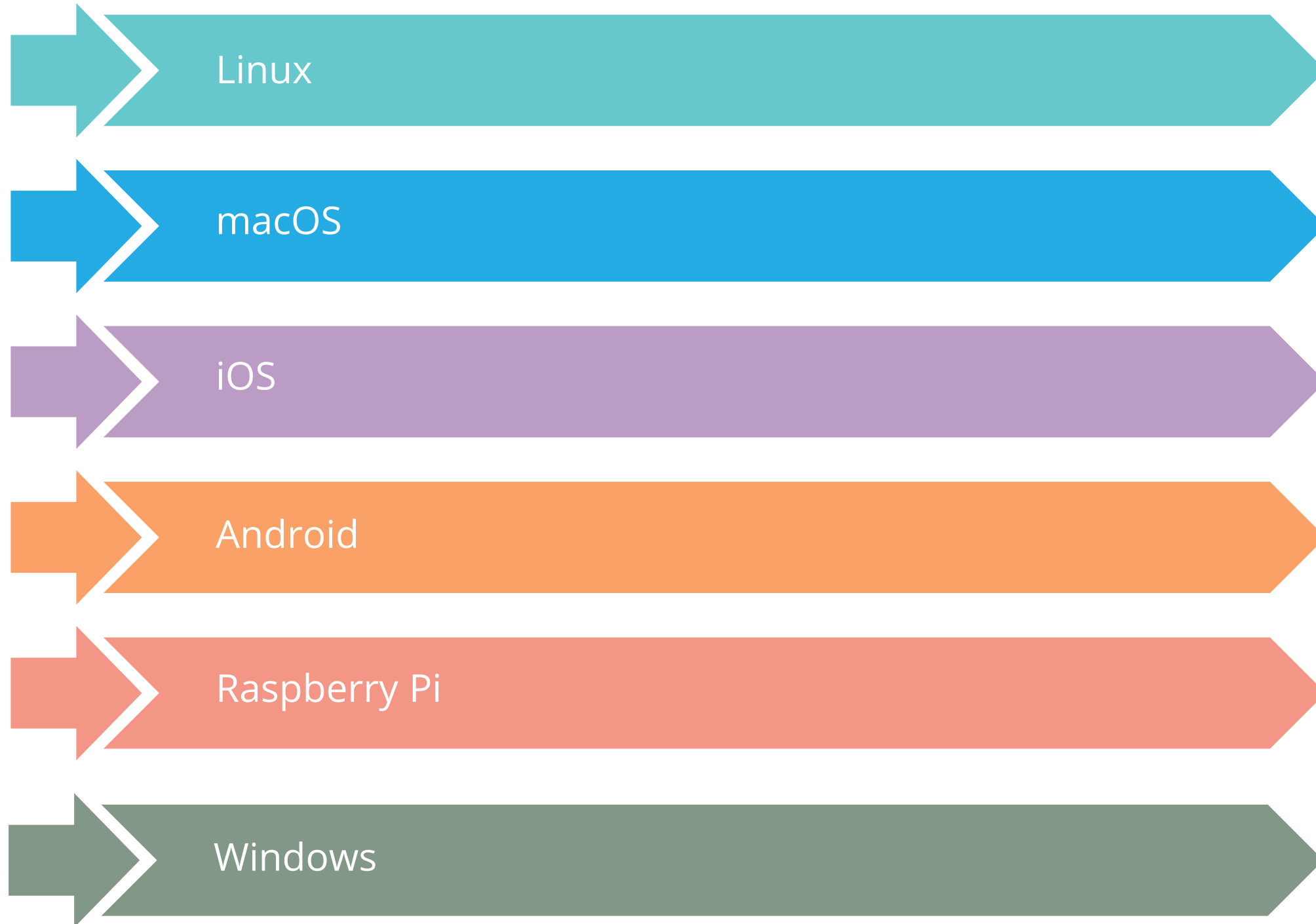
TensorFlow: Parallel Computation

TensorFlow utilizes parallel computation techniques to efficiently process data and accelerate the training of deep learning models.



TensorFlow: Multiple Environment Friendly

TensorFlow is compatible with desktop and mobile software environments like:



TensorFlow: Open-Source Platform

TensorFlow is supported by a vibrant and an extensive community of users and developers.

- Is one of the most popular open-source projects on GitHub
- Has a dedicated team of passionate and helpful developers
- Has a growing community, contributing to improve it

TensorFlow: High-Level Abstraction

High-level abstractions in TensorFlow simplify the process of creating deep learning models by providing:

Simplified model development

Faster prototyping

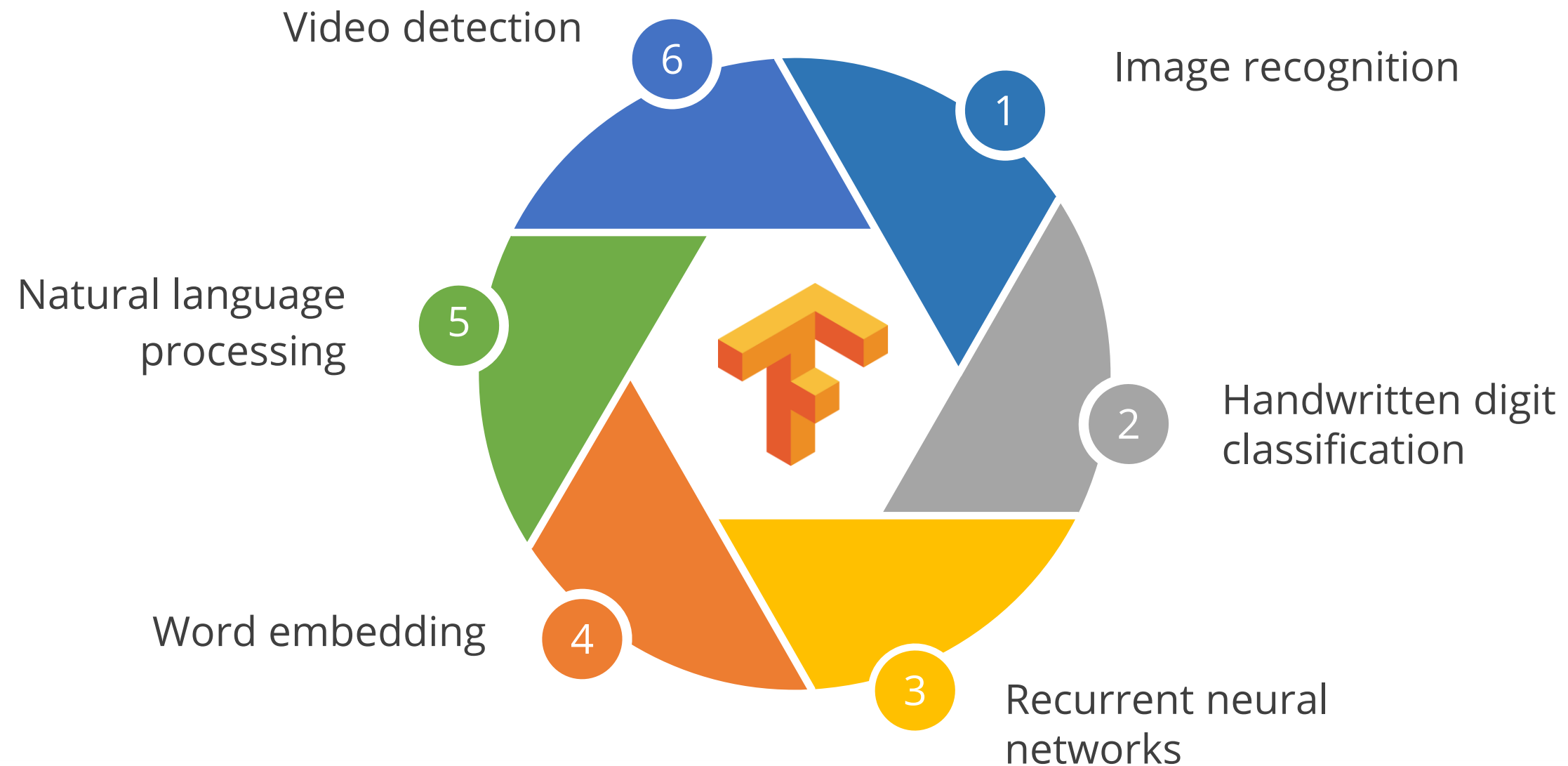


Increased productivity

Seamless experimentation

Applications of TensorFlow

TensorFlow can train and run deep neural networks for:



Companies Using TensorFlow

Top companies leveraging TensorFlow are:

Uber

The Uber logo, consisting of the word "Uber" in a bold, black, sans-serif font.

SAP



DeepMind



Coca-Cola

The Coca-Cola logo, featuring the words "Coca-Cola" in a red, cursive script font.

Dropbox



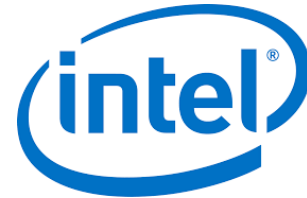
Companies Using TensorFlow

Top companies leveraging TensorFlow are:

eBay



Intel



Qualcomm



Google



Use-cases: Healthcare

TensorFlow's capabilities in processing and analyzing medical images can significantly aid in diagnosing diseases, with more speed and precision.



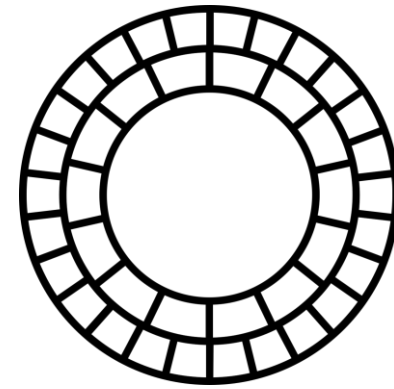
Google developed DermAssist, using TensorFlow, which allows one to take pictures of the skin and make possible diagnoses for various skin conditions

Use-cases: Social Media

TensorFlow has been implemented in the following applications.



X implemented TensorFlow to rank tweets according to user preferences.



VSCO, a photo-sharing app, uses TensorFlow to suggest filters for photos.



RankBrain, a search engine released by Google, uses TensorFlow to process search results.

Use-cases: Education

TensorFlow is used to filter toxic chat messages in classrooms in a virtual learning platform.



It is also used to accurately identify a student's current capabilities and help them decide the most suitable course of action in the future.

Use-cases: Retail

Quite a few e-commerce platforms use TensorFlow to get personalized customer recommendations.



Cosmetics companies use TensorFlow to create an augmented reality experience for customers to test makeup on their faces.

Assisted Practice



Let us understand the concept of Tensors and training DNN with TensorFlow using Jupyter Notebooks.

- 5.02_Introduction to Tensors
- 5.03_Hands-on with TensorFlow: Part A
- 5.04_Training DNN Using TensorFlow

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic



Installation of TensorFlow

Prerequisites for Installing TensorFlow

The following system requirements must be met before installing TensorFlow:

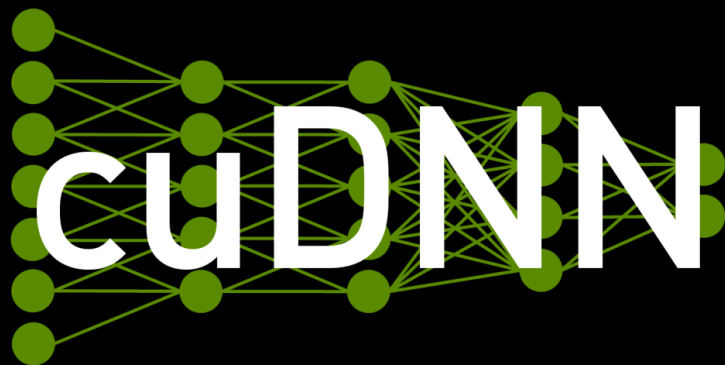
- ✓ Ubuntu 18.04 or higher
- ✓ macOS 10.15 or higher
- ✓ Windows 10 or higher
- ✓ Python 3.8 or higher

Setting Up the System

Windows users can install Anaconda and follow these steps:

1

Link to install TensorFlow:
<https://www.tensorflow.org/install/source#gpu>



It has all the versions of the TensorFlow library, along with the versions of the cuDNN and CUDA libraries; the latter are important for GPU support in TensorFlow.

It also has a Python version that is compatible with the given TensorFlow versions.

Setting Up the System

2

Link to install CUDA:
<https://developer.nvidia.com/cuda-toolkit-archive>



NVIDIA
CUDA

Download and install the right version of the
NVIDIA CUDA toolkit

Example: To install TF 2.5 or above, install CUDA
11.2

Setting Up the System

3

To download the library, create an account with NVIDIA

Create Your Account

Email

xyz@gmail.com

Password


.....

Fair

Confirm password

.....

☒ Stay logged in [Log In With Security Device](#)

☐ I am human  [Privacy - Terms](#)

By proceeding, I agree to the [NVIDIA Account Terms Of Use](#) and [Privacy Policy](#).

Create Account

[More Signup Options](#)

Use the following link to create an account:
https://nvidia.custhelp.com/app/utils/create_account

Download the cuDNN library

Setting Up the System

4

Extract the content from the downloaded zip file and copy it

Name	Type
 bin	File folder
 include	File folder
 lib	File folder

Setting Up the System

5 In **Program Files**, look for the folder **NVIDIA GPU Computing Toolkit**

 NVIDIA Corporation	17-07-2023 14:42	File folder
 NVIDIA GPU Computing Toolkit	17-07-2023 13:00	File folder

Setting Up the System

6

Go to the **CUDA** folder and paste the cuDNN files to the folder **v11.2**

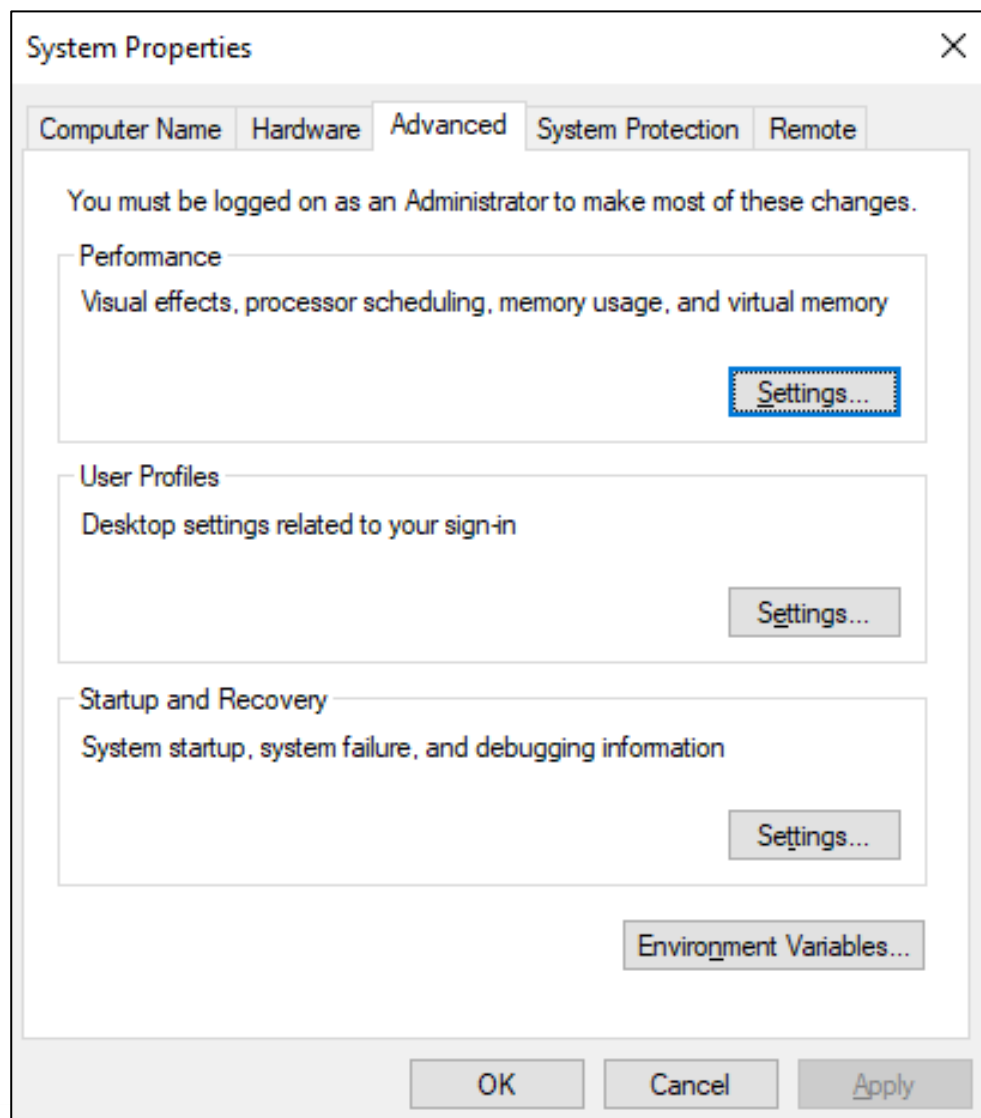
s PC > OS (C:) > Program Files > NVIDIA GPU Computing Toolkit > CUDA > v11.2				
Name	Date modified	Type	Size	
bin	17-07-2023 14:43	File folder		
compute-sanitizer	17-07-2023 14:43	File folder		
extras	17-07-2023 14:43	File folder		
include	17-07-2023 14:43	File folder		
lib	17-07-2023 14:43	File folder		
libnvvp	17-07-2023 14:43	File folder		
nvml	17-07-2023 14:43	File folder		
nvvm	17-07-2023 14:43	File folder		
nvvm-prev	17-07-2023 14:43	File folder		
src	17-07-2023 14:43	File folder		
tools	17-07-2023 14:43	File folder		
CUDA_Toolkit_Release_Notes	01-12-2020 15:54	Text Document	47 KB	
DOCS	01-12-2020 15:54	File	1 KB	
EULA	01-12-2020 15:54	Text Document	62 KB	
README	01-12-2020 15:54	File	1 KB	

Inside the folder NVIDIA GPU Computing Toolkit

Setting Up the System

7

Go to the **bin folder** and copy the entire path to that location

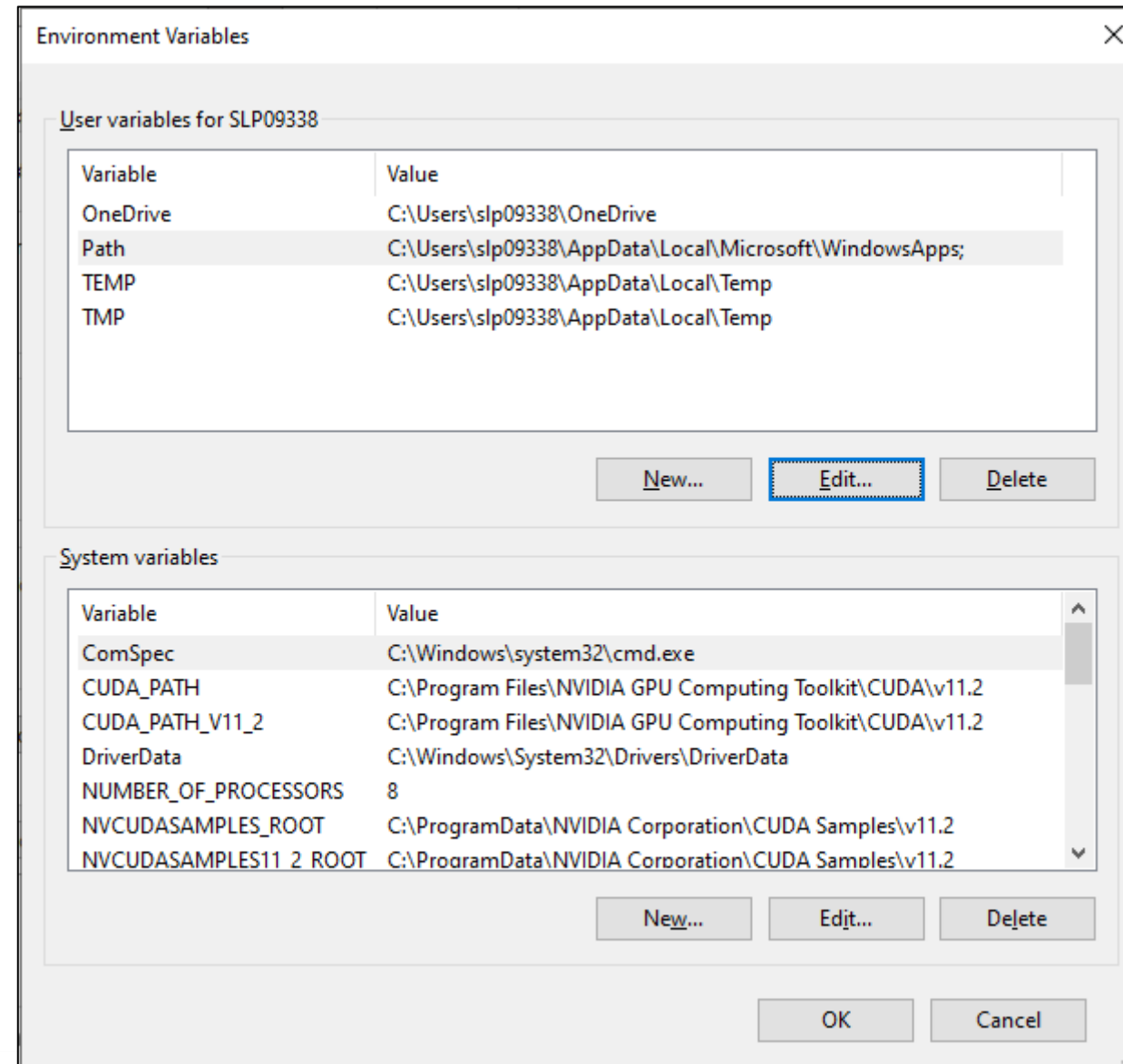


Next, open **System Properties**
and go to **Environment
Variables**

Setting Up the System

8

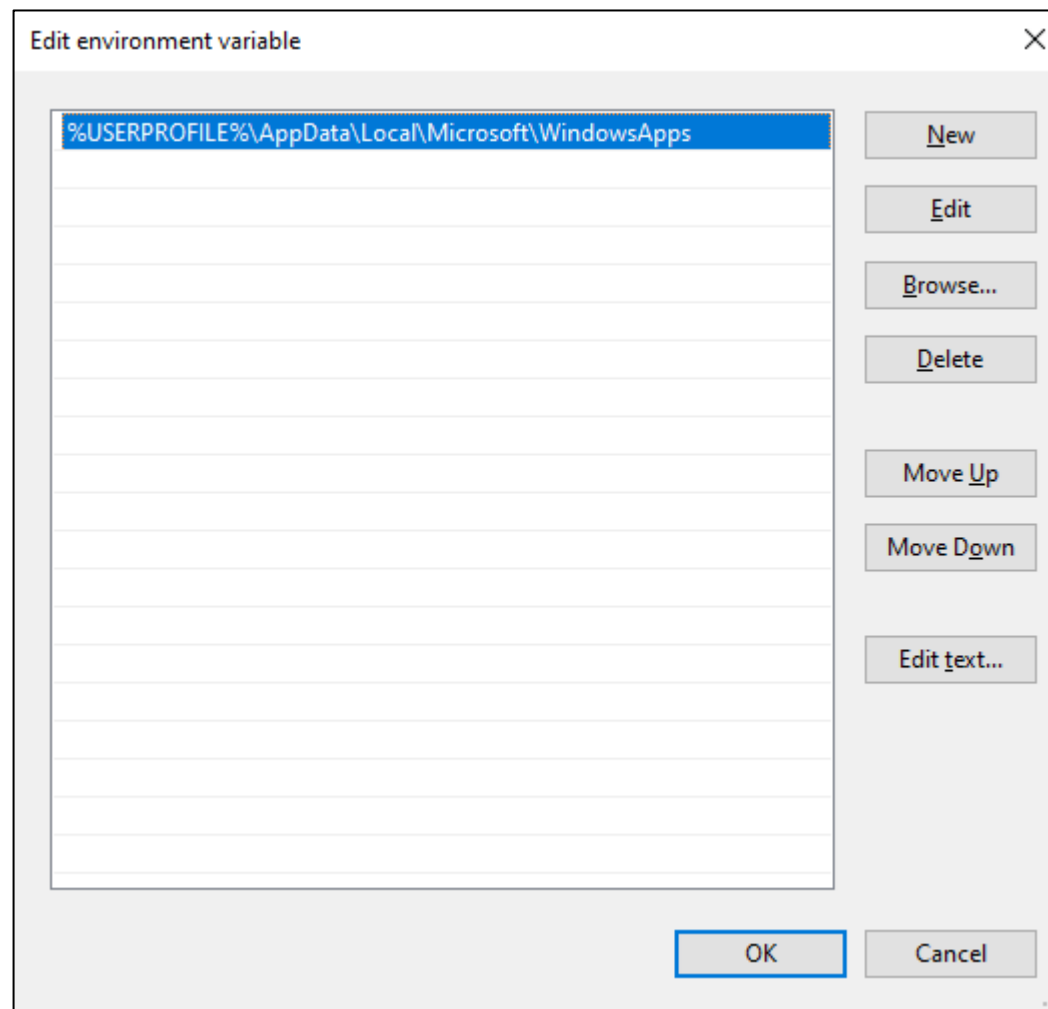
Double-click on the **Path** here



Setting Up the System

9

Click on **New**


















Paste the copied path or location at the bottom and press **OK**

Setting Up the System

10

Go back, open the **libnvvp** folder, and copy its path

Name	Date modified	Type	Size
 bin	17-07-2023 14:43	File folder	
 compute-sanitizer	17-07-2023 14:43	File folder	
 extras	17-07-2023 14:43	File folder	
 include	17-07-2023 14:43	File folder	
 lib	17-07-2023 14:43	File folder	
 libnvvp	17-07-2023 14:43	File folder	
 nvml	17-07-2023 14:43	File folder	
 nvvm	17-07-2023 14:43	File folder	
 nvvm-prev	17-07-2023 14:43	File folder	
 src	17-07-2023 14:43	File folder	
 tools	17-07-2023 14:43	File folder	
 CUDA_Toolkit_Release_Notes	01-12-2020 15:54	Text Document	47 KB
 DOCS	01-12-2020 15:54	File	1 KB
 EULA	01-12-2020 15:54	Text Document	62 KB
 README	01-12-2020 15:54	File	1 KB

Setting Up the System

11

Open **System Properties** and go to **Environment Variables**

Double-click on this path, click on **New**, and then paste the copied path.

Once all the steps are completed, restart the computer before installing TensorFlow.

Installing TensorFlow

TensorFlow can be installed by a single line of code. Once installed, run the following command in the Python interpreter to ensure successful installation.

Code:

```
pip install tensorflow

import tensorflow as tf
print(tf.__version__)
```

Output:

```
[1]: import tensorflow as tf
      print(tf.__version__)

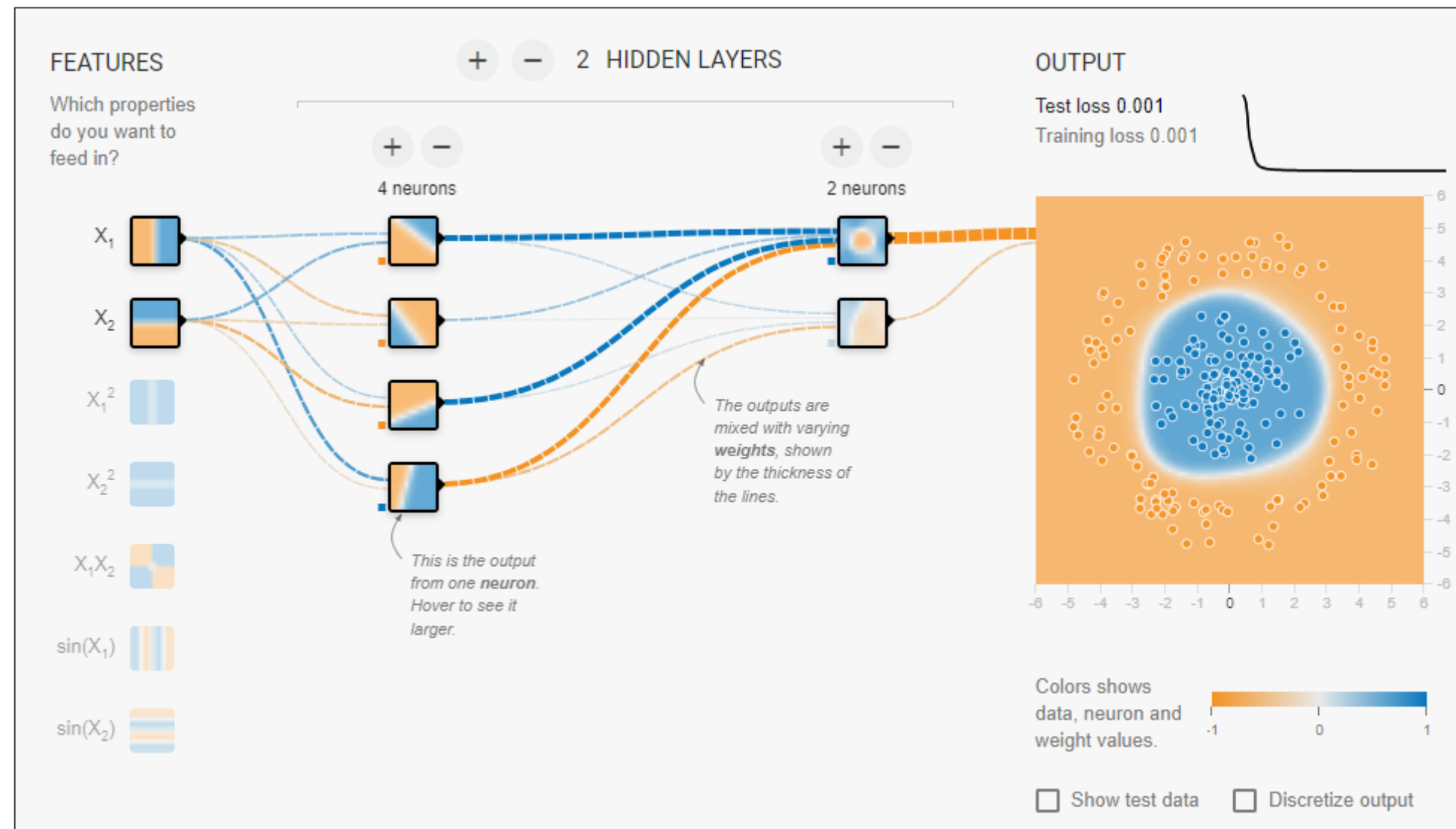
      2.8.0
```



TensorFlow Playground




TensorFlow Playground

TensorFlow Playground is a browser-based application for learning about and experimenting with neural networks. It can be used to visualize how hyperparameter changes influence a machine learning model.



Hands-on with the TensorFlow Playground

Step 1: Login to the website





			Epoch 000,000	Learning rate 0.03	Activation Tanh	Regularization None	Regularization rate 0	Problem type Classification
---	--	---	------------------	-----------------------	--------------------	------------------------	--------------------------	--------------------------------

Hands-on with the TensorFlow Playground

Step 2: Set the dataset for the ratio of training to test data, noise, and batch size

DATA

Which dataset do you want to use?



Ratio of training to test data: 50%




Noise: 0

Batch size: 10

REGENERATE

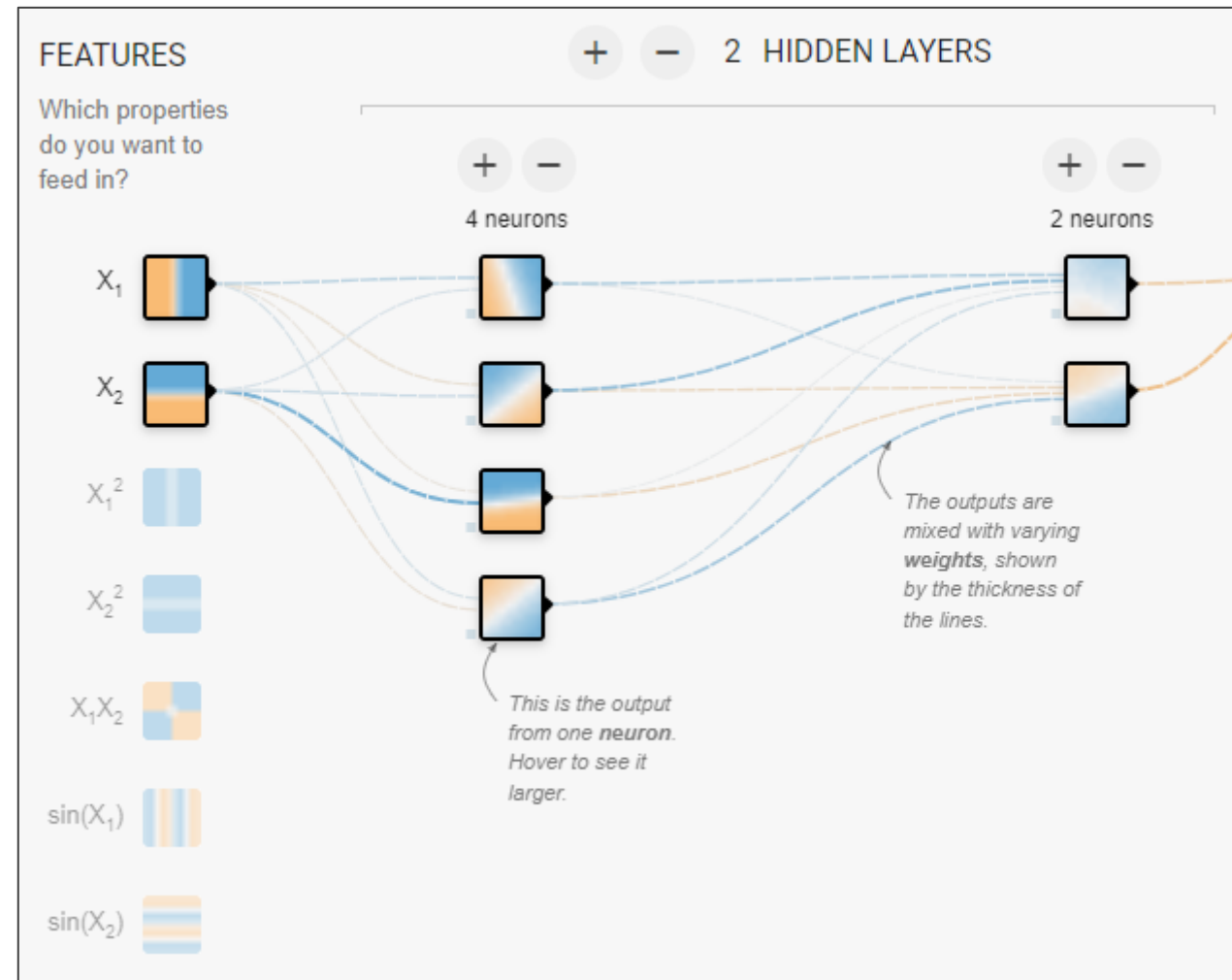
Hands-on with the TensorFlow Playground

Step 3: Set the features for the Epoch, Learning rate, Activation, Regularization, Regularization rate, and Problem type

			Epoch 000,000	Learning rate 0.03	Activation Sigmoid	Regularization L1	Regularization rate 0.001	Problem type Classification
--	--	---	------------------	-----------------------	-----------------------	----------------------	------------------------------	--------------------------------




Hands-on with the TensorFlow Playground

Step 4: Set the fully connected dense layer






Hands-on with the TensorFlow Playground

Step 5: Click on the play button to start the training

			Epoch 000,000	Learning rate 0.03	Activation Sigmoid	Regularization L1	Regularization rate 0.001	Problem type Classification
---	---	---	------------------	-----------------------	-----------------------	----------------------	------------------------------	--------------------------------

Hands-on with the TensorFlow Playground

Step 6: Click the play button to pause the timer once the test loss reaches its lowest value



Epoch
000,000

Learning rate
0.03

Activation
Sigmoid

Regularization
L1

Regularization rate
0.001

Problem type
Classification

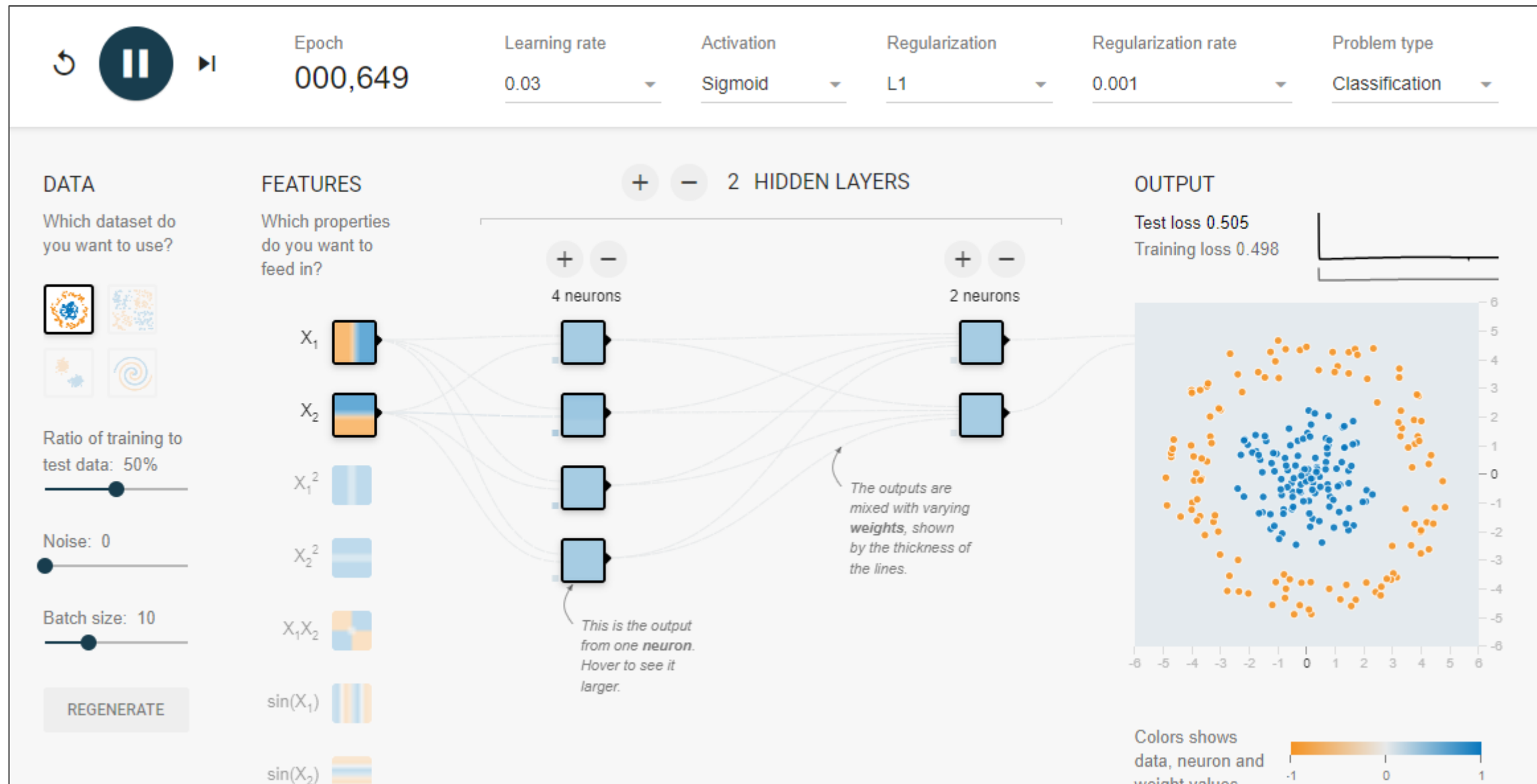
OUTPUT

Test loss 0.505
Training loss 0.498



Hands-on with the TensorFlow Playground

Result: It can be observed that within the 649 epoch, a decision boundary is built between two classes.





TFLearn

What Is TFLearn?

TFLearn is a modular and transparent deep learning library built on TensorFlow. It provides a high-level API to facilitate and speed up experiments while maintaining full transparency and compatibility with TensorFlow.

Features of TFLearn



Easy to use, understand, and implement



Fast prototyping through highly modular built-in components



Full transparency over TensorFlow



Powerful helper functions to train any TensorFlow graph



Easy and clear graph visualization



Effortless device placement for utilizing multiple CPUs or GPUs

Comparing TFLearn with TensorFlow

The difference between TFLearn and TensorFlow are:

Feature	TensorFlow	TFLearn
Level of API	Lower-level, allowing control over architecture	High-level, focusing on user-friendliness and rapid prototyping
Scalability	Supports CPUs, GPUs, TPUs, and distributed training	Limited to TensorFlow's underlying scalability
Deployment options	Robust deployment on various platforms including edge devices	Dependent on TensorFlow's deployment options
Best suited for	Complex and large-scale projects, advanced users	Quick experimentation and beginners and intermediate users
Integration and visualization	Integrates with TensorBoard and other advanced tools	Integrates with TensorFlow and is simpler without advanced tools

TFLearn Installation

The easiest way to install TFLearn is to run the following command using pip:

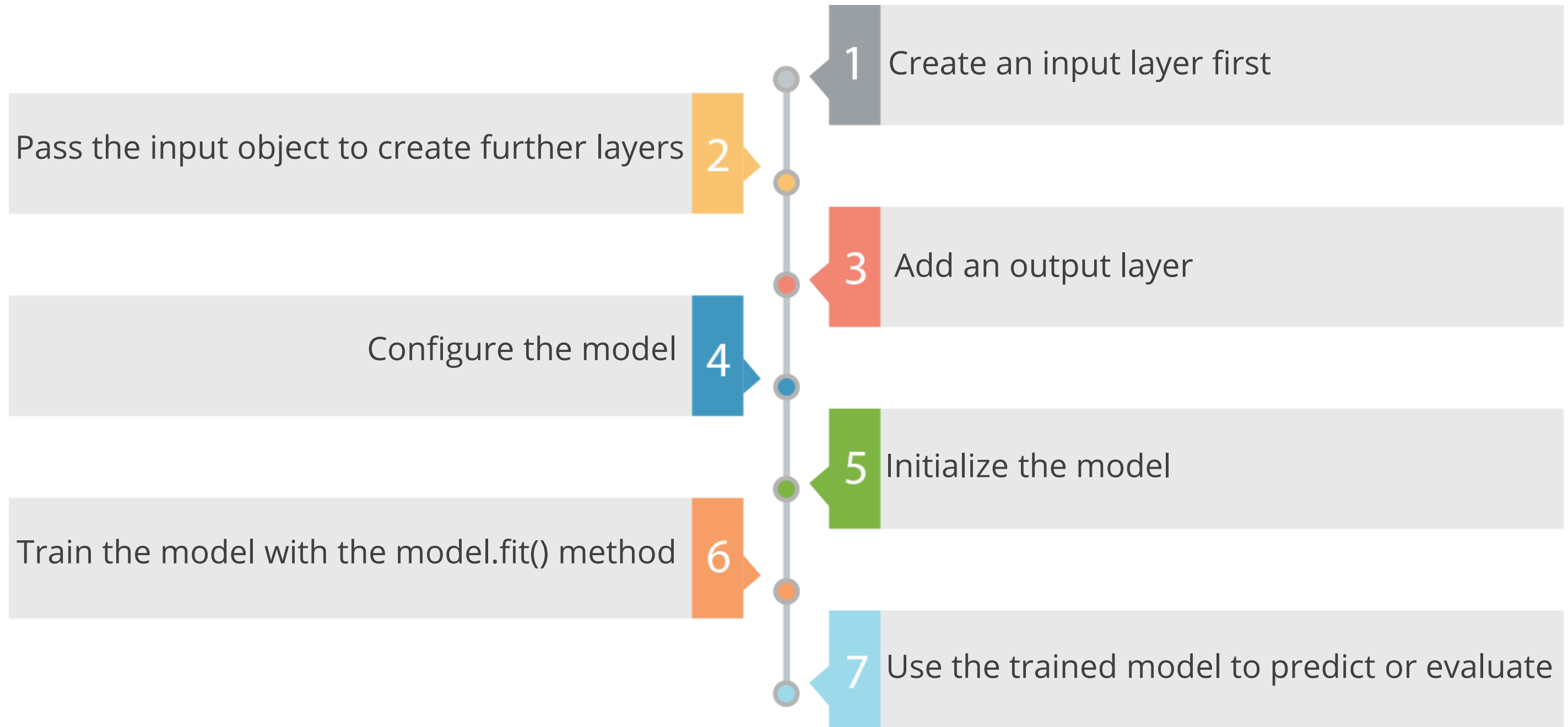


For the latest stable version:

- `pip install tflearn`

Workflow of TFLearn

TFLearn API workflow:



Layers of TFLearn

Layers are a core concept of TFLearn. TFLearn brings layers that represent an abstract set of operations to make building neural networks more convenient.

Here is a list of all currently available layers:

File	Layers
core	input_data, fully_connected, dropout, custom_layer, reshape, flatten, activation, single_unit, highway, one_hot_encoding, time_distributed
conv	conv_2d, conv_2d_transpose, max_pool_2d, avg_pool_2d, upsample_2d, conv_1d, max_pool_1d, avg_pool_1d, residual_block, residual_bottleneck, conv_3d, max_pool_3d, avg_pool_3d, highway_conv_1d, highway_conv_2d, global_avg_pool, global_max_pool
recurrent	simple_rnn, lstm, gru, bidirectional_rnn, dynamic_rnn
embedding	embedding
normalization	batch_normalization, local_response_normalization, l2_normalize
merge	merge, merge_outputs
estimator	regression

Built-In Operations of TFLearn

Besides *layers*, TFLearn also provides multiple different operations to be used when building a neural network.

Here is a list of all the currently available operations:

File	Operations
activations	linear, tanh, sigmoid, softmax, softplus, softsign, relu, relu6, leaky_relu, prelu, elu
objectives	softmax_categorical_crossentropy, categorical_crossentropy, binary_crossentropy, mean_square, hinge_loss, roc_auc_score, weak_cross_entropy_2d
optimizers	SGD, RMSProp, Adam, Momentum, AdaGrad, Ftrl, AdaDelta
metrics	Accuracy, Top_k, R2
initializations	zeros, uniform, uniform_scaling, normal, truncated_normal, xavier, variance_scaling
losses	l1, l2

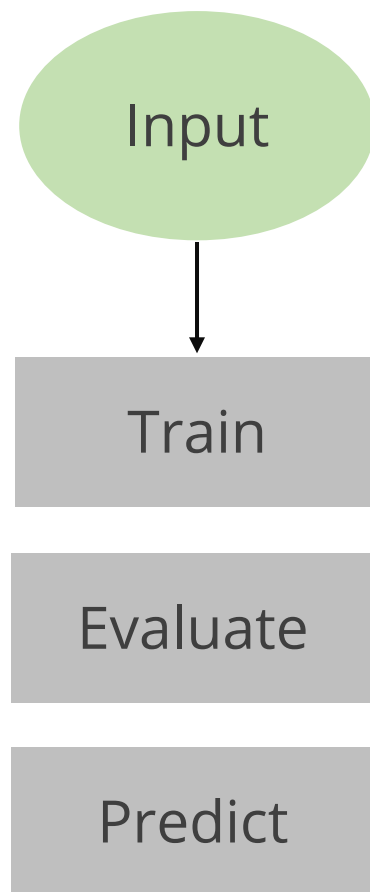
Training of TFLearn

Training functions are another core feature of TFLearn. TensorFlow has no prebuilt APIs to train a network, so TFLearn integrates a set of functions that can easily handle any neural network training, for any number of inputs, outputs, and optimizers.



Trainer, Evaluator, and Predictor

In TFLearn, the terms called *trainer*, *evaluator*, and *predictor* refer to specific roles or functionalities within the framework that handles different aspects of working with neural network models.



- Any TensorFlow graph can be trained using the *helpers* functions offered by TFLearn.
- By introducing real-time monitoring, batch sampling, moving averages, TensorBoard logs, and other data input methods, it is possible to significantly enhance the convenience of the training process.
- It accepts any quantity of inputs, outputs, and optimization operations.

Trainer, Evaluator, and Predictor

TFLearn creates a **TrainOp** class to describe an optimization procedure such as backprop. Here is how it is defined:

Syntax:-

```
import tensorflow as tf
import tflearn

#Define your network architecture
input_placeholder = tf.placeholder(tf.float32, shape=[None, input_size])
target_placeholder = tf.placeholder(tf.float32, shape=[None, num_classes])
my_network = tflearn.fully_connected(input_placeholder, 32)
loss = tflearn.objectives.categorical_crossentropy(my_network, target_placeholder)
accuracy = tflearn.metrics.accuracy(my_network, target_placeholder)

#Create TrainOp and Trainer
trainop = tflearn.TrainOp(net=my_network, loss=loss, metric=accuracy)
model = tflearn.Trainer(train_ops=trainop, tensorboard_dir='/tmp/tflearn')
```

Trainer, Evaluator, and Predictor

TrainOps can be fed into a **Trainer** class that will handle the whole training process, considering all TrainOps together as a whole model.

Syntax:-

```
#Train the model
model.fit(feed_dicts={input_placeholder: X, target_placeholder: Y},
n_epoch=10, batch_size=128, show_metric=True)
```


Trainer, Evaluator, and Predictor

TFLearn models are useful for more complex models to handle multiple optimization.

Syntax:-

```
#Create TrainOp objects for each training operation
trainop1 = tflearn.TrainOp(net=network1, loss=loss1)
trainop2 = tflearn.TrainOp(net=network2, loss=loss2)
trainop3 = tflearn.TrainOp(net=network3, loss=loss3)

#Create Trainer with multiple TrainOps
model = tflearn.Trainer(train_ops=[trainop1, trainop2, trainop3])

#Train the model with different feed dictionaries for each training operation
feed_dict1 = {in1: X1, label1: Y1}
feed_dict2 = {in2: X2, in3: X3, label2: Y2}
model.fit(feed_dicts=[feed_dict1, feed_dict2])
```

Trainer, Evaluator, and Predictor

For prediction, TFLearn implements an **Evaluator** class that works the same as the trainer. It takes a parameter and returns the predicted value.

Syntax:-

```
#Create Evaluator
model = tflearn.Evaluator(network)

#Make predictions
predictions = model.predict(feed_dict={input_placeholder: X})
```

Trainer, Evaluator, and Predictor

The **Trainer** class in TFLearn utilizes the **is_training** boolean variable to handle network behavior during training, testing, and prediction.

Syntax:-

```
#Example for Dropout:
x = ...

def apply_dropout(): #Function to apply when training mode ON.
    return tf.nn.dropout(x, keep_prob)

is_training = tflearn.get_training_mode() #Retrieve is_training variable.
tf.cond(is_training, apply_dropout, lambda: x) #Only apply dropout at training time.
```

Trainer, Evaluator, and Predictor

To make it easy, TFLearn implements functions to retrieve that variable or change its value.

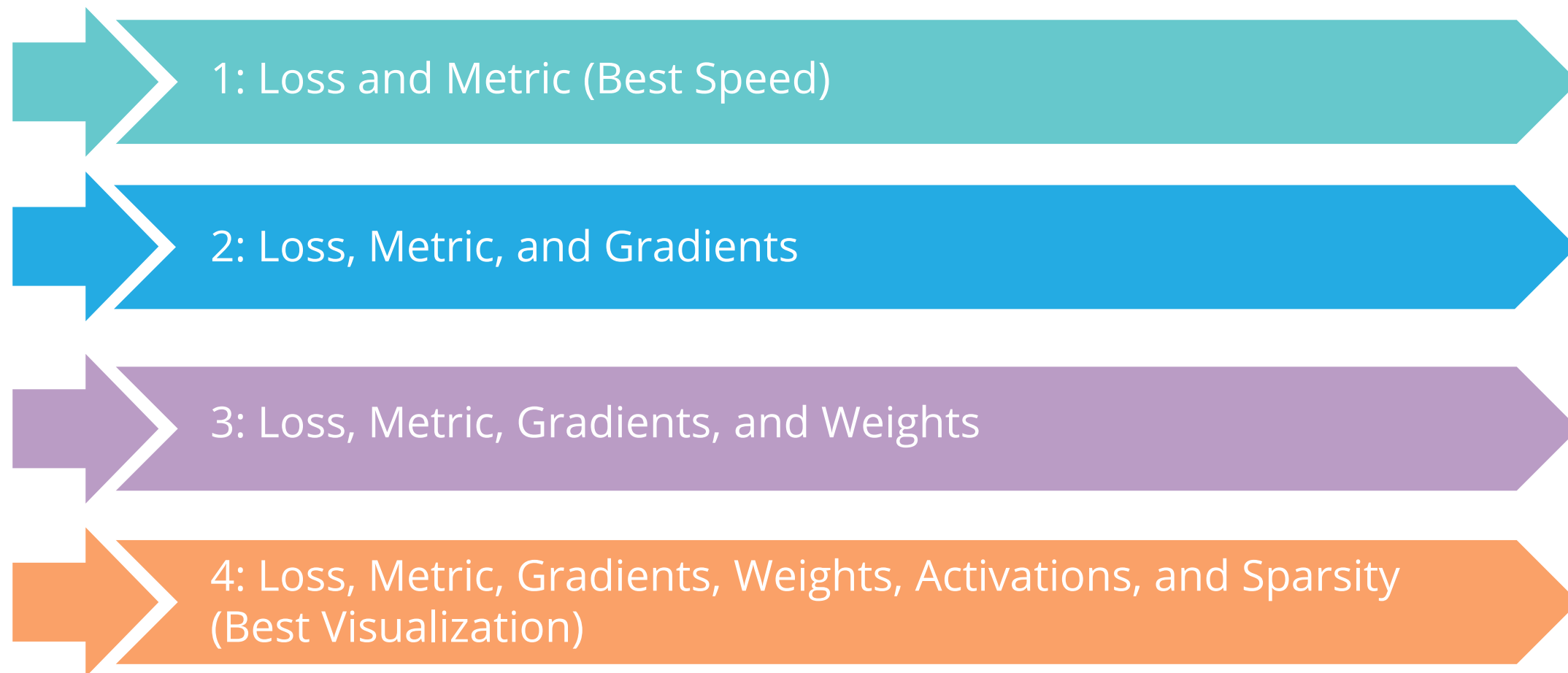
Syntax:-

```
#Set training mode ON (set is_training var to True)
tflearn.is_training(True)

#Set training mode OFF (set is_training var to False)
tflearn.is_training(False)
```

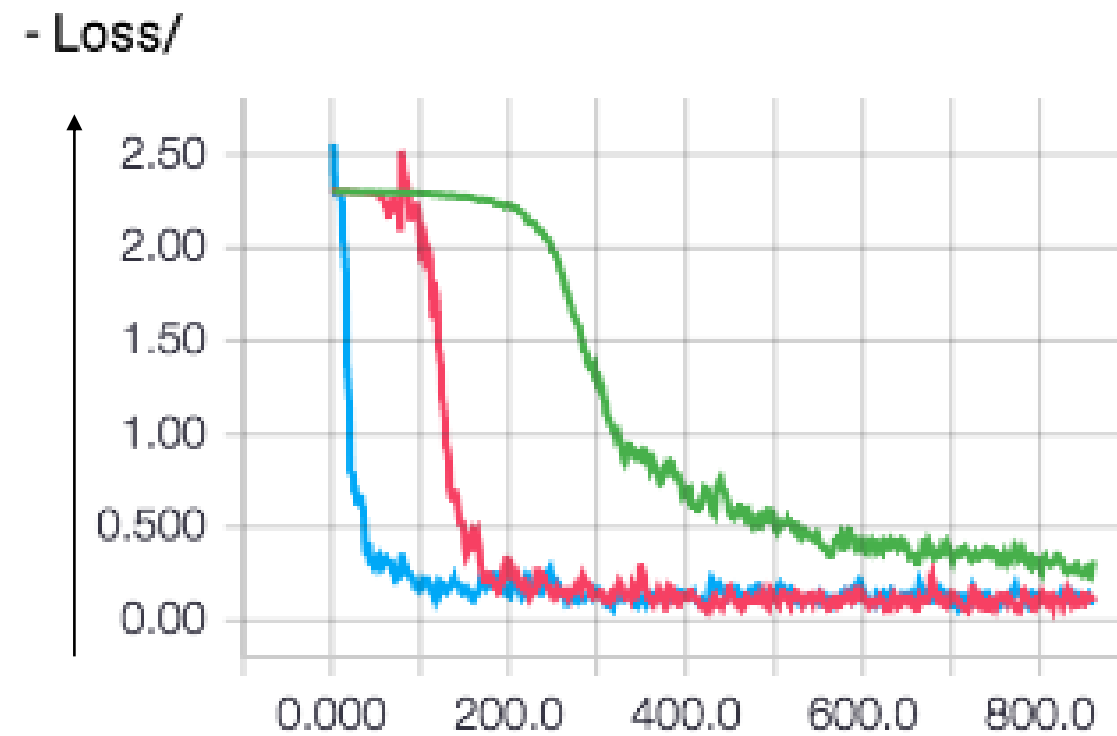
Visualization

TFLearn handles the creation and management of logs for training metrics without requiring manual setup. Currently, it supports a *verbose level* to automatically manage summaries like:

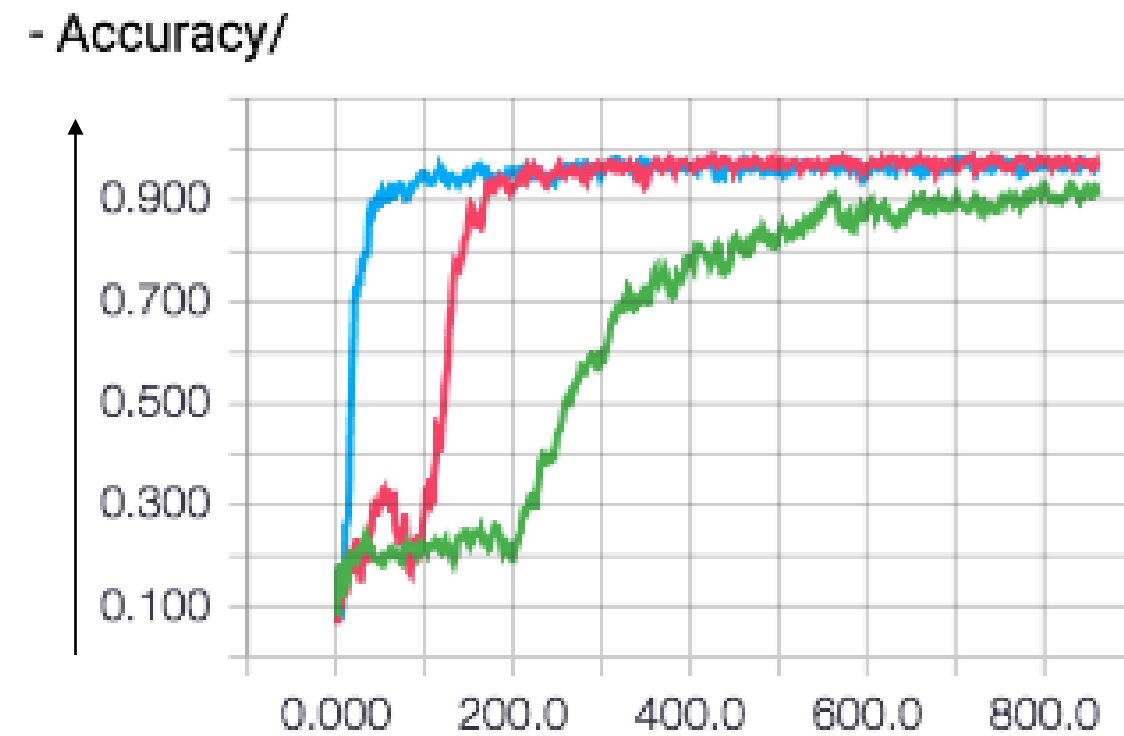


Visualization: Loss and Accuracy

Visualizing loss and accuracy aids in analyzing and optimizing neural network training.



Epochs

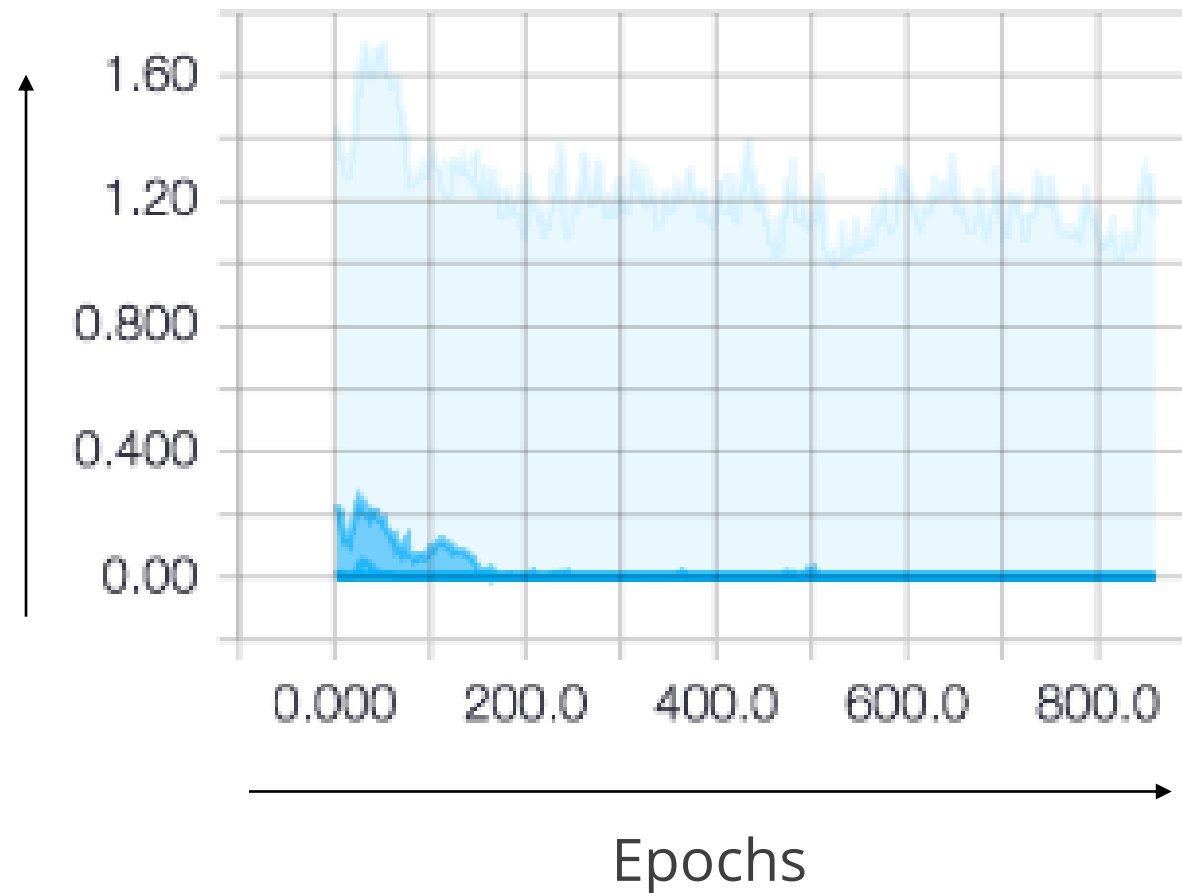


Epochs

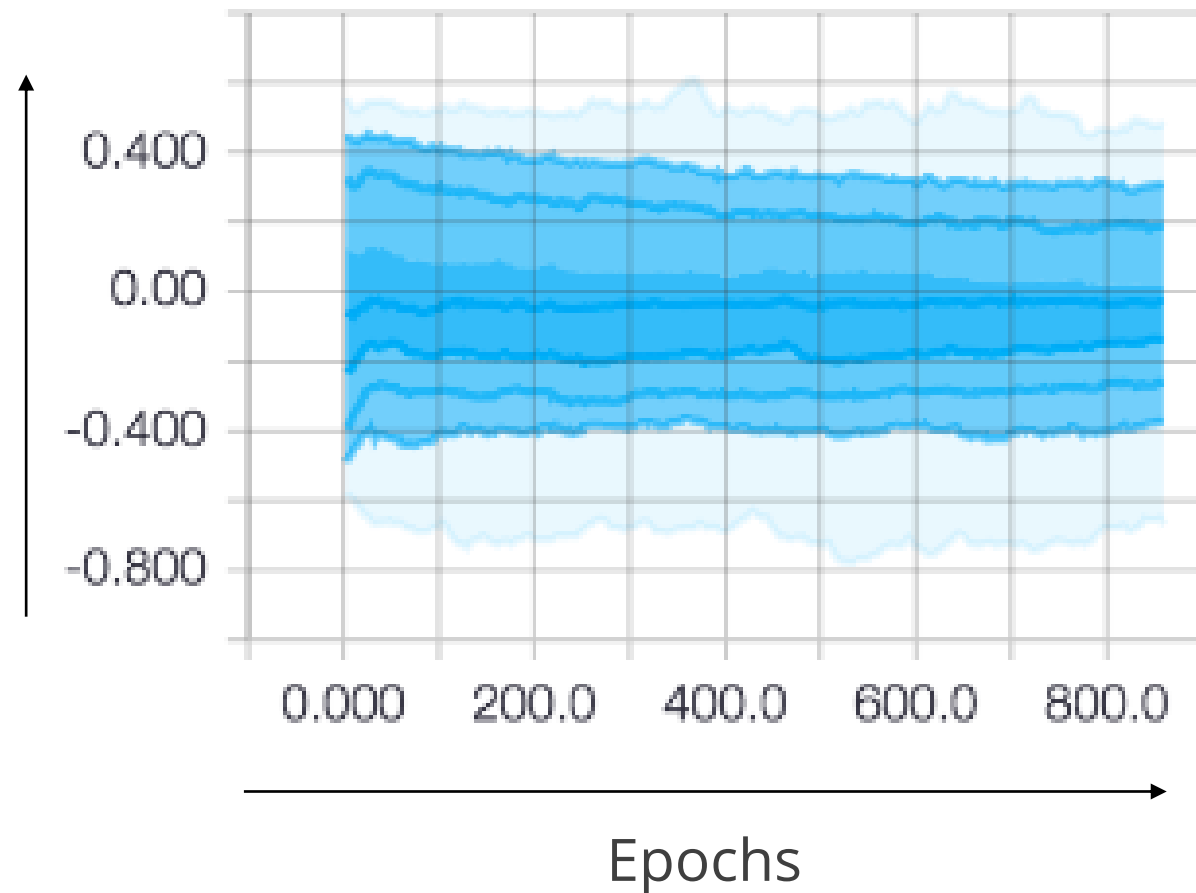
Visualization: Layers

Visualizing the layers between convolutional operations and convolutional weight layers provides insights into feature extraction and transformations within a convolutional neural network.

Conv2D/Relu/Activations



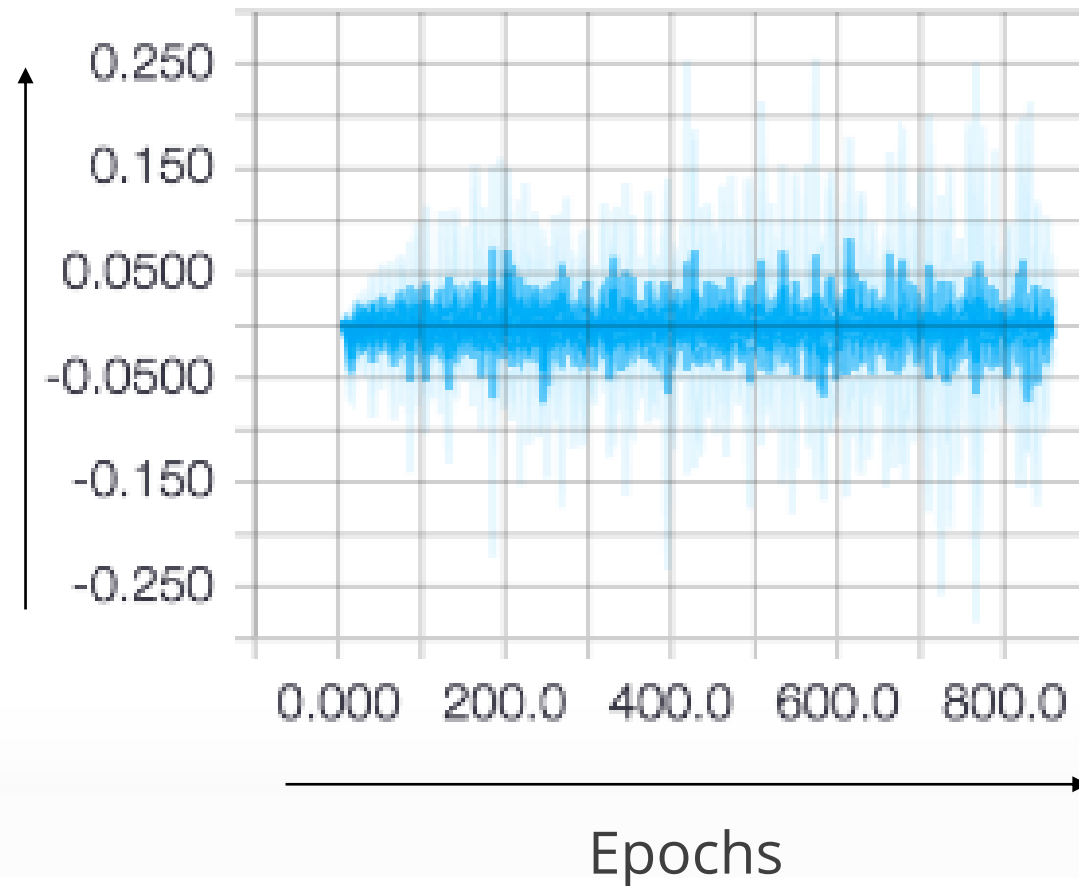
Conv2D/W



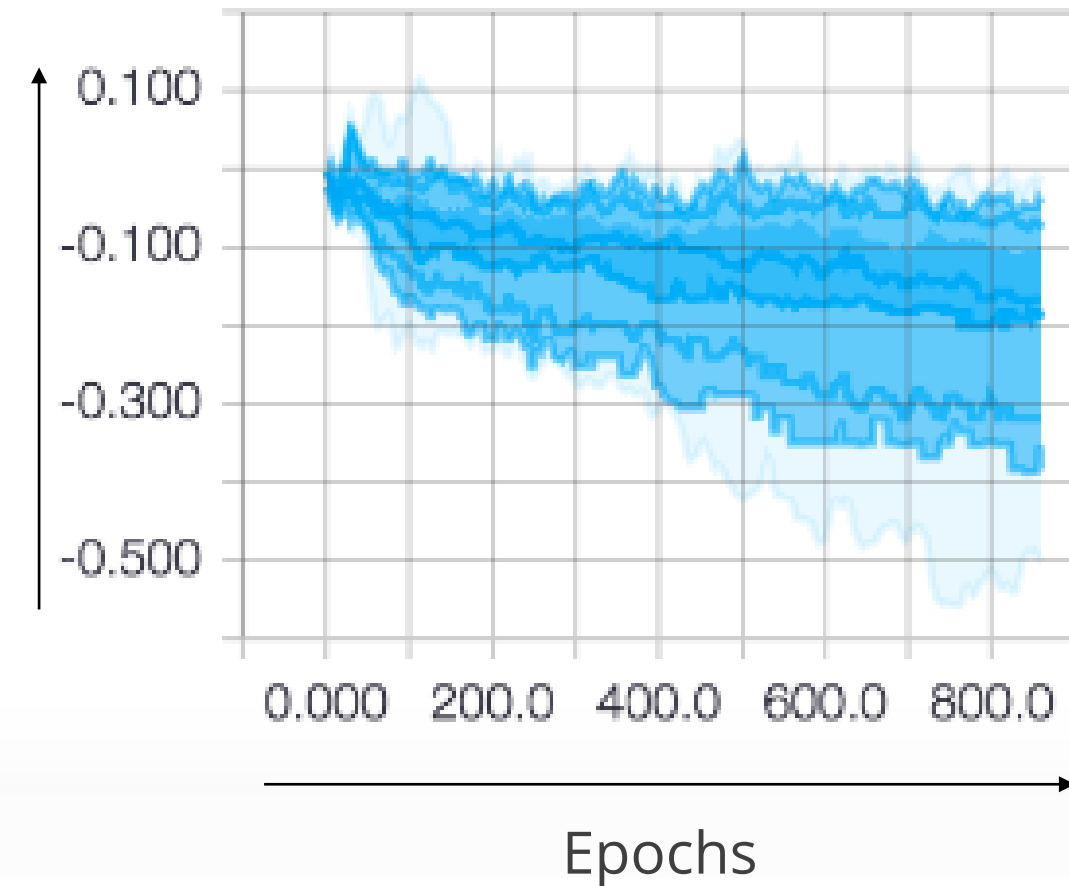
Visualization: Layers

Visualizing the layers between convolutional weight gradients and convolutional bias layers unveils the influence of biases on feature extraction within a convolutional neural network.

Conv2D/W/Gradients/



Conv2D/b





Introduction to Keras

Keras

Keras is a high-level deep learning API that simplifies the process of building and training neural network models, providing a user-friendly interface and extensive support for various deep learning tasks. It is:



Fast



Easy to implement



Modular in nature

What Is Keras?

Is a high-level neural network
API, written in Python

Is most powerful and easy to use
for developing and evaluating
deep learning models



Runs seamlessly on CPU and GPU

Keras

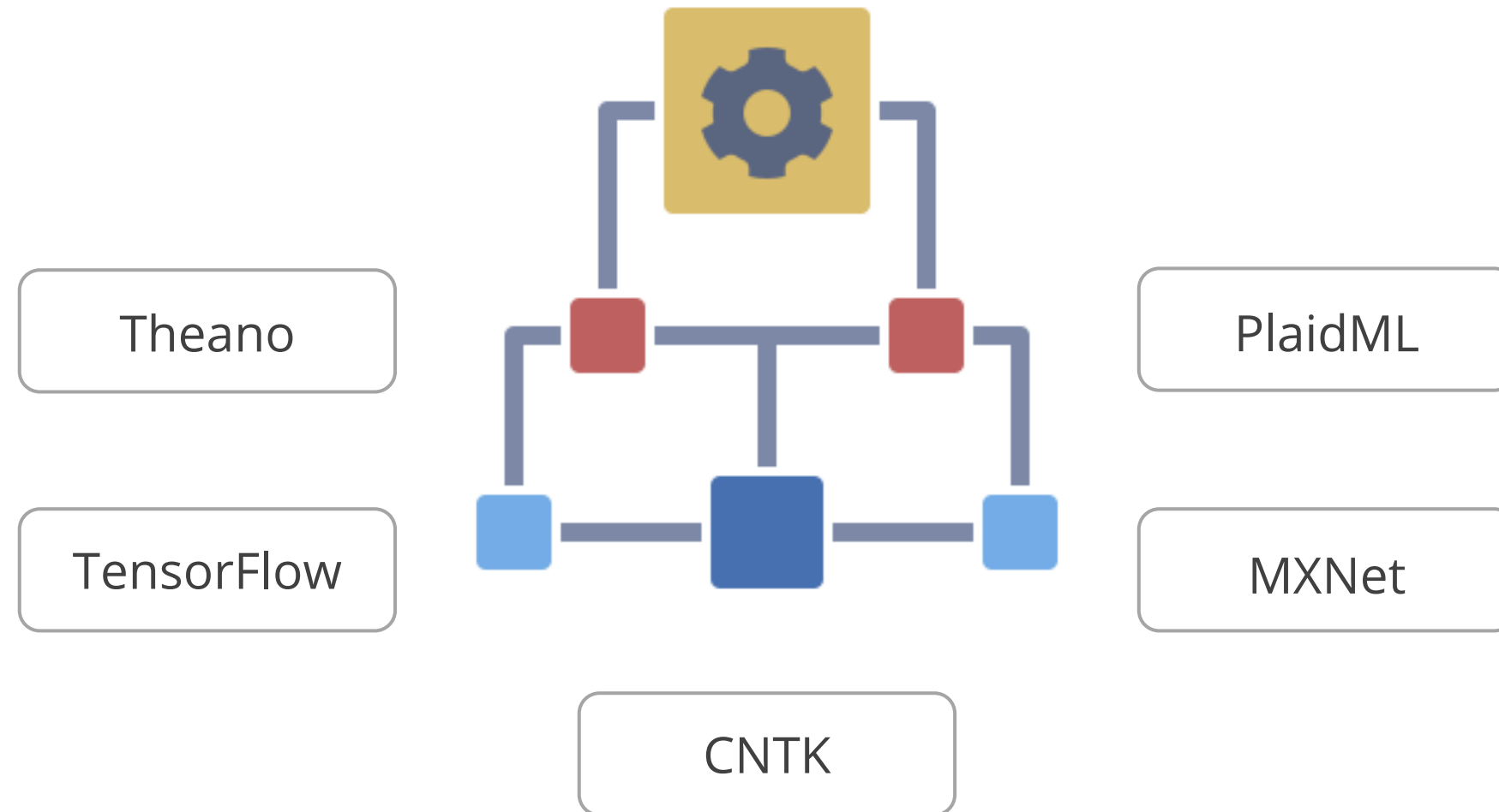
Keras is closely tied to the TensorFlow library and acts as an interface for it.



It also allows the user to define and train neural network models with only a few lines of code.

Framework Supported by Keras

The frameworks supported by Keras are:



Keras: Features



Time-Saving

Is efficient as a versatile library, enabling utilization across a wide range of machine learning tasks and accommodating multiple models



Flexible

Follows the principle of progressive disclosure of complexity



Powerful

Provides industry-strength performance and scalability

Additionally, it enables the user to define and train neural network models with very little code.

Keras: Backends

Keras uses TensorFlow, Theano, MxNet, and CNTK (Microsoft) as backends.

Keras

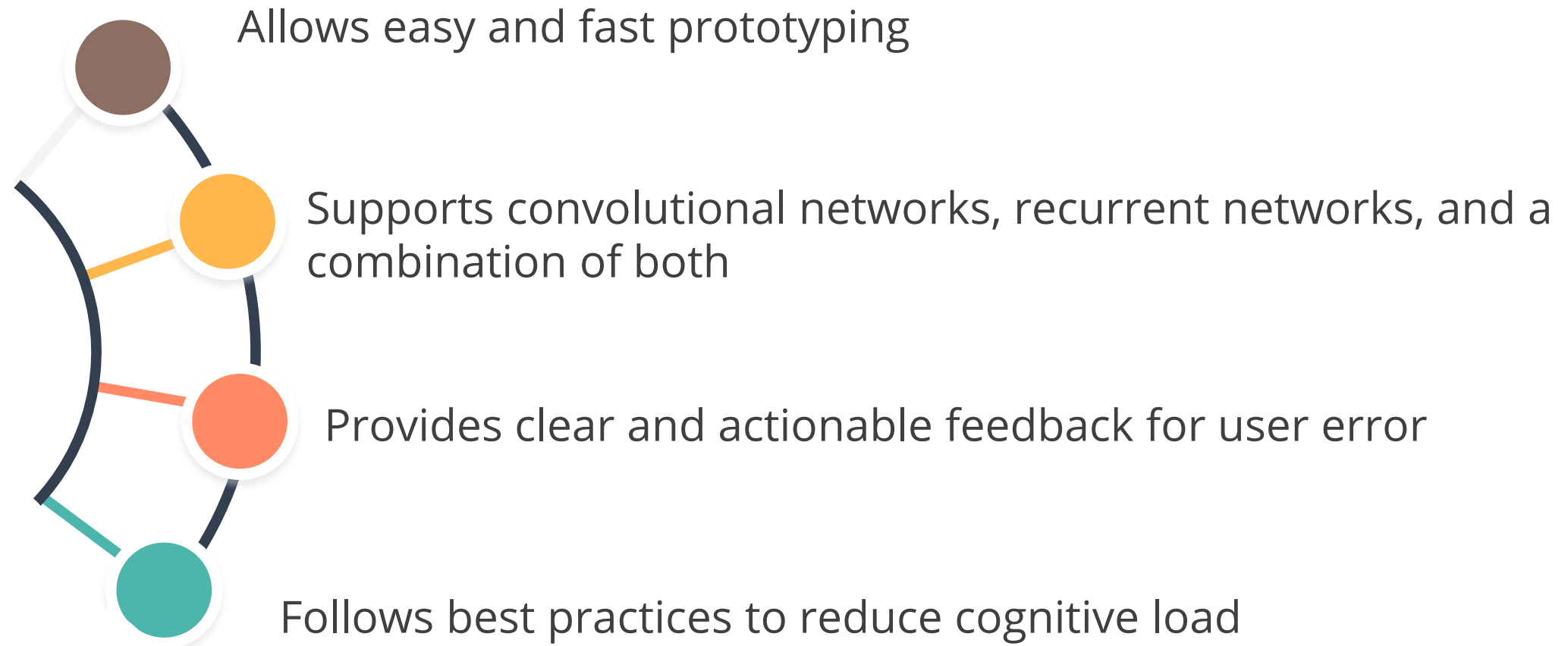
TensorFlow, MxNet, CNTK, Theano

CPU

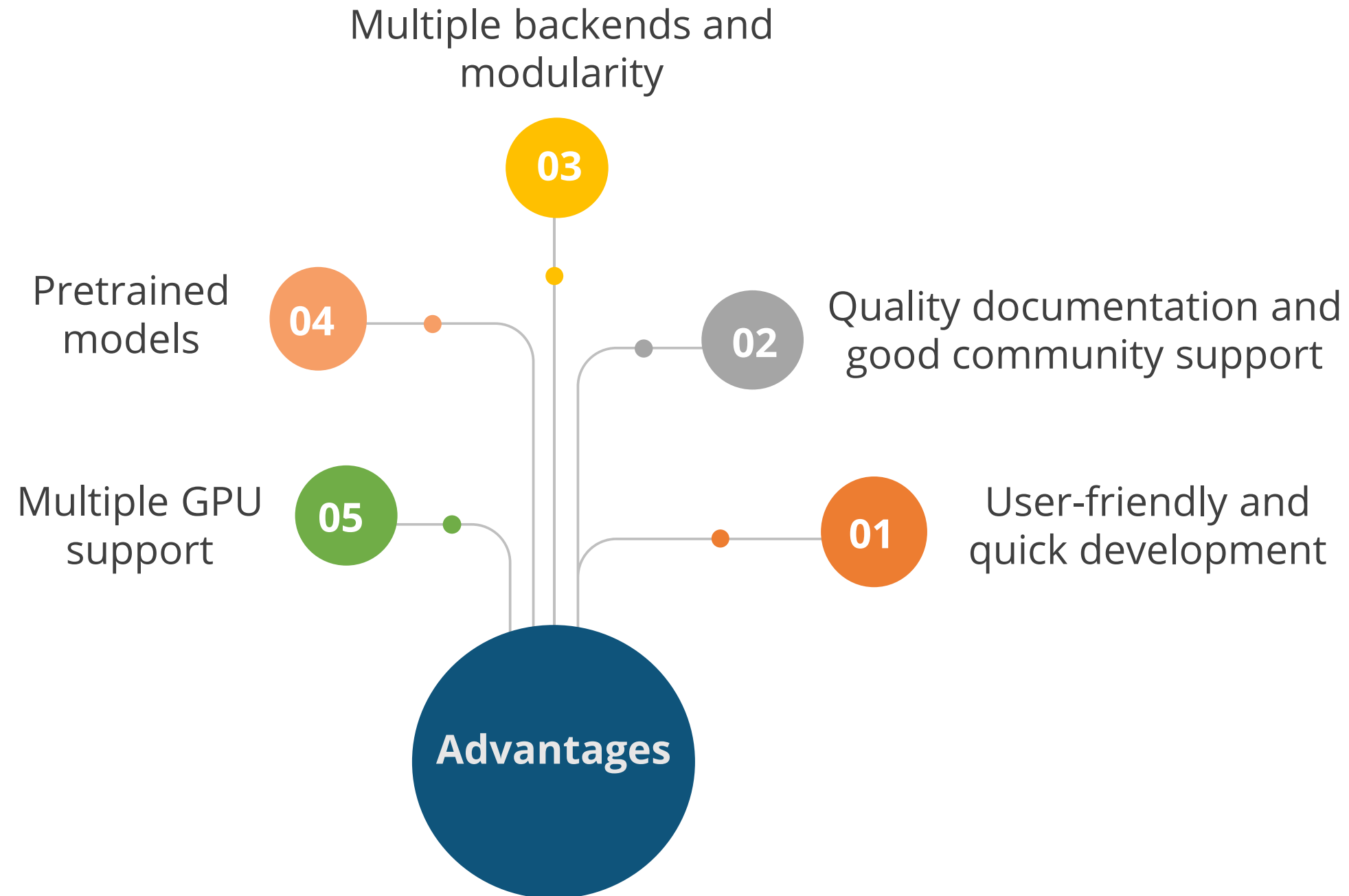
GPU

TPU

Why Use Keras?



Advantages of Keras



Advantages of Keras

User-friendly and quick development

Keras provides clear and concise APIs. It enables quick experimentation, that allows for easy and quick development of deep learning models.

Advantages of Keras

Quality documentation and good community support

Keras is well-documented, with a wide range of tutorials and resources available. It also has a huge and active community support with several open-source platforms.

Advantages of Keras

Multiple backends and modularity

One can train a Keras model on one backend and test its results on another. Its modular design lets a user plug together building blocks with limited restriction.

Advantages of Keras

Pretrained models

Keras provides access to various deep learning models with pretrained weights; one can use these to make predictions or extract features.

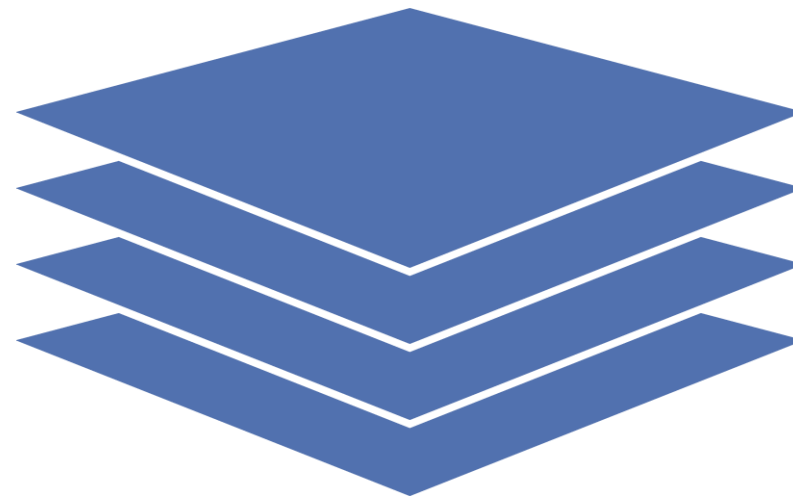
Advantages of Keras

Multiple GPU support

Keras allows one to train the model on a single GPU or use multiple GPUs. It also provides built-in support for data parallelism and can process a large amount of data.

Keras API Components: Layers

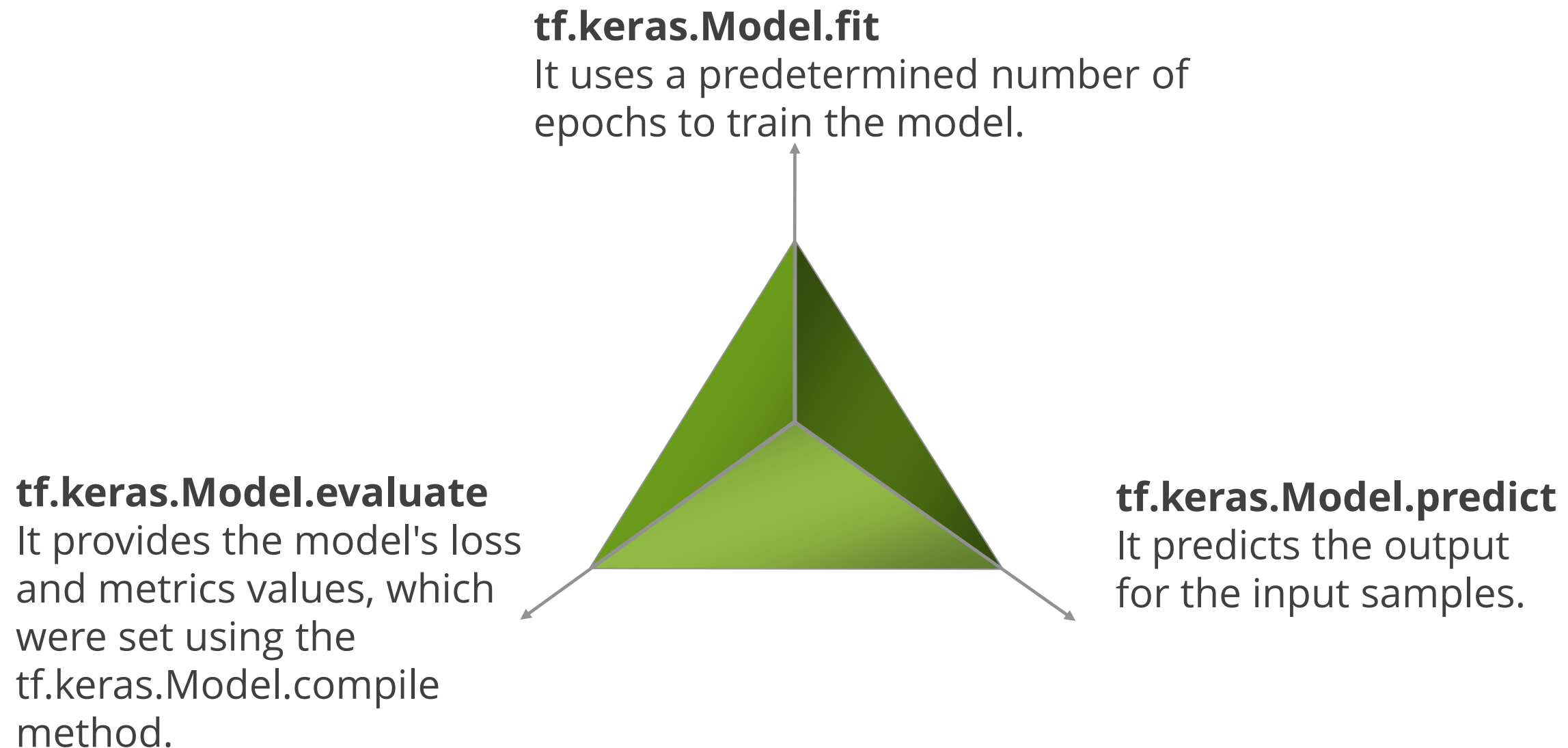
The `tf.keras.layers.Layer` class is the fundamental abstraction in Keras. A layer encapsulates a state (weights) and some computation.



Layers are recursively composable. If a layer instance is assigned as an attribute of another layer, the outer layer will start tracking the weights created by the inner layer.

Keras API Components: Models

A model is an object that combines layers and can be trained on data.
The `tf.keras.Model` class features built-in training and evaluation methods such as:



Sequential and Functional API in Keras

Keras framework provides two primary ways to build models: Sequential API and Functional API.

Sequential API: The Sequential API is the simpler of the two and is best suited for models that have a single input tensor and output tensor. It allows one to create models layer-by-layer in a linear stack.

Functional API: It is more flexible and powerful, allowing the creation of complex models, such as multi-input or multi-output models, models with shared layers, and models with non-sequential data flows.

Sequential API

It is more straightforward to use, making it ideal for building linear stacks of layers where each layer has one input and output tensor.

Example:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,
Activation

model = Sequential([
    Dense(64, input_shape=(input_dim,),
activation='relu'),
    Dense(10, activation='softmax')
])
```

Functional API

It provides more flexibility and allows you to create models with complex architectures, including multiple inputs and outputs, shared layers, and skip connections.

Example:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense,
Activation

input_layer = Input(shape=(input_dim,))
hidden_layer = Dense(64, activation='relu')(input_layer)
output_layer = Dense(10,
activation='softmax')(hidden_layer)

model = Model(inputs=input_layer, outputs=output_layer)
```

Sequential API vs. Functional API

Aspect	Sequential API	Functional API
Architecture	Linear stack of layers	Customized and complex
Complexity	Limited to simple models	Supports complex architectures
Flexibility	Limited customization	High customization and control
Usage	Simple models, quick start	Complex architectures, control

Assisted Practice



Let us understand the concept of sequential and functional APIs in TensorFlow and Keras using Jupyter Notebooks

- 5.09_Sequential_APIs_in_TensorFlow
- 5.10_Functional_APIs_in_TensorFlow

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

Creating a Keras Model

The following are the steps to create a Keras model:

Step 1

Import the required libraries and load the dataset

Step 2

Assign number of layers, number of nodes in the layers, and the activation function to be used

Step 3

Compile the loss function and evaluate a set of weights

Step 4

Fit the model using backpropagation and weight optimization with input data

Creating a Keras Model

The following are the steps to create a Keras model:

Step 5

Evaluate the model's performance on a separate validation dataset

Step 6

Predict the output with the prepared model

Step 1: Import the libraries

Import statements are used to import specific classes, load the dataset, and set image dimensions.

Syntax:-

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D, MaxPooling2D, Flatten,
Dense
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Image dimensions
img_width, img_height = x_train.shape[1], x_train.shape[2]
```


Step 2: Create the Model

The sequential model is a linear stack of layers.

Syntax:-

```
# Model definition
model = Sequential()
model.add(Convolution2D(16, (5, 5), activation='relu', input_shape=(img_width,
img_height, 3)))
model.add(MaxPooling2D(2, 2))
model.add(Convolution2D(32, (5, 5), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

The above-mentioned code creates a sequential model in Keras with convolutional, pooling, and dense layers, ultimately forming a network for image classification with 10 classes.

Step 3: Compile the Model

- The loss function evaluates a set of weights.
- The optimizer searches through different weights for the network and optional metrics to collect and report during training.
- Set **metrics=['accuracy']** for the classification problem.

Syntax:-

```
# Compile the model
model.compile(loss='binary_crossentropy',
              optimizer='adam', metrics=['accuracy'])
```

This syntax compiles the model with binary cross-entropy loss, Adam optimizer, and accuracy as the metrics for evaluation.

Step 4: Fit the Model

This syntax trains the model on the training data, displays the progress, and evaluates the performance using validation data.

Syntax:-

```
# Train the model
model.fit(x_train, y_train,
          batch_size=32,
          epochs=10,
          verbose=1,
          validation_data=(x_test, y_test))
```

- Executes a model for some data
- Trains and iterates data in batches

Output:

```
Epoch 9/10
1563/1563 [=====] - 45s 29ms/step - loss: 0.2357 - accuracy: 0.9187 - val_loss: 1.4263 - val_accuracy: 0.6640
Epoch 10/10
1563/1563 [=====] - 45s 29ms/step - loss: 0.1857 - accuracy: 0.9362 - val_loss: 1.6172 - val_accuracy: 0.6651
```

Step 5: Evaluate the Model

This syntax evaluates the trained model on the test data and prints the test loss and test accuracy.

Syntax:-

```
# Evaluate the model  
score = model.evaluate(x_test, y_test,  
verbose=0)  
print('Test loss:', score[0])  
print('Test accuracy:', score[1])
```

Output:

```
Test loss: 1.4093151092529297  
Test accuracy: 0.6583999991416931
```

Step 6: Predict with the Model

The syntax makes predictions on the test data, **x_test**, using the trained model and returns the predicted classes.

Syntax:-

```
classes=model.predict(x_test,batch_size=128)
```

Output:

```
79/79 [=====] - 2s 26ms/step
```

Implementation of Loss Function

The loss function's task is to estimate the model's error or loss and change the weights in the hidden layers. This reduces the loss in the next assessment.

Loss functions in TensorFlow are implemented using TensorFlow's computational graph and automatic differentiation capabilities.

How Are Loss Functions Implemented in TensorFlow?

To implement a loss function, a choice of loss function should first be made so that it fits the framing of the specific predictive modeling issue.

The output layer's configuration must be sufficient for the loss function used.

Syntax:

```
model.compile(optimizer=tf.optimizers.Adam(),  
              loss='mae',  
              metrics='mean_absolute_error')
```

Assisted Practice



Let us understand the topics of TensorFlow and Keras, using Jupyter Notebook.

- 5.11_Hands-on TensorFlow and Keras: Part B

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

Key Takeaways

- TensorFlow is a flexible, open-source library for machine learning and artificial intelligence.
- TensorBoard, a suite of visualization tools, makes it easier to visualize the computational graph.
- TFLearn is a high-level deep learning library built on TensorFlow, providing simplified development of neural networks with a user-friendly API.
- Keras is a high-level deep learning API designed for easy and fast model development, built on top of TensorFlow.





Knowledge Check

Knowledge Check

1

Which of the following tasks can be performed using TensorFlow?

- A. Image recognition
- B. Natural language processing
- C. Video detection
- D. All of the above



Knowledge Check

1

Which of the following tasks can be performed using TensorFlow?

- A. Image recognition
- B. Natural language processing
- C. Video detection
- D. All of the above



The correct answer is **D**

TensorFlow can be used for image recognition, natural language processing, and video detection.

Knowledge Check

2

Which of the following defines TensorFlow?

- A. It is a flexible open-source library for machine learning and artificial intelligence.
- B. It is a closed source library for deep learning and image recognition.
- C. It is a programming language for web development.
- D. It is a database management system.



Knowledge Check

2

Which of the following defines TensorFlow?

- A. It is a flexible open-source library for machine learning and artificial intelligence.
- B. It is a closed source library for deep learning and image recognition.
- C. It is a programming language for web development.
- D. It is a database management system.



The correct answer is **A**

TensorFlow is a flexible open-source library for machine learning and artificial intelligence.

Knowledge Check

3

What is the sequential API in TensorFlow?

- A. It is a method of creating complex models with multiple inputs or outputs
- B. It is a way to define models, layer by layer
- C. It is a library for natural language processing
- D. It is a type of optimization algorithm



Knowledge Check

3

What is the sequential API in TensorFlow?

- A. It is a method of creating complex models with multiple inputs or outputs
- B. It is a way to define models, layer by layer
- C. It is a library for natural language processing
- D. It is a type of optimization algorithm



The correct answer is **B**

Sequential API is a way to define models, layer by layer.



Thank You!