Deep Learning

TensorFlow

Learning Objectives

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

- Learn the process of installation of TensorFlow 2.8.0
- Examine the TensorFlow sequential and functional Application Programming Interface (APIs)
- Create a model with TensorFlow
- Construct an image recognition model
- Analyze the basics of 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 stay ahead of the competition in the fast-paced technology industry.

Introduction to TensorFlow

Discussion

Discussion: TensorFlow

Duration: 10 minutes

- What is TensorFlow?
- What are the features of 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.

What Is TensorFlow?

A popular open-source library for deep learning

Developed by the Google Brain team and released in 2015

TensorFlow™

Used mainly for classification, perception, understanding, discovering, prediction, and creation

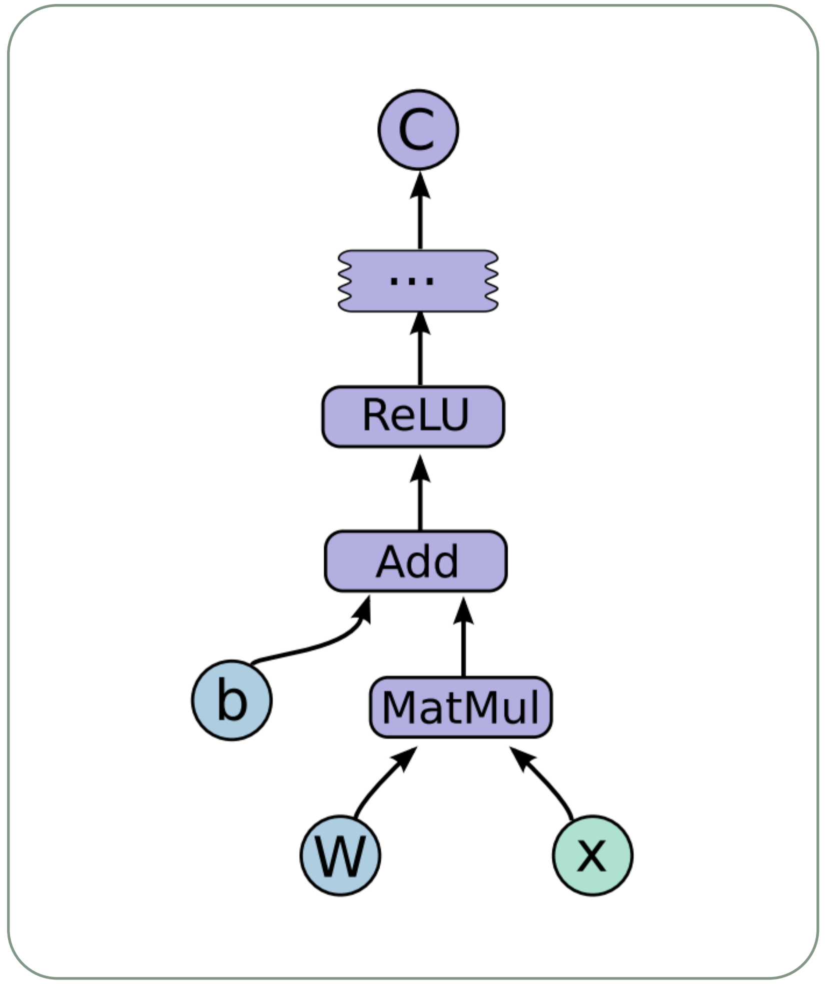
Why Is TensorFlow Necessary?

TensorFlow operates on a system of data flow graphs, which allows for efficient computation and parallel processing, 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 using data flow graphs.



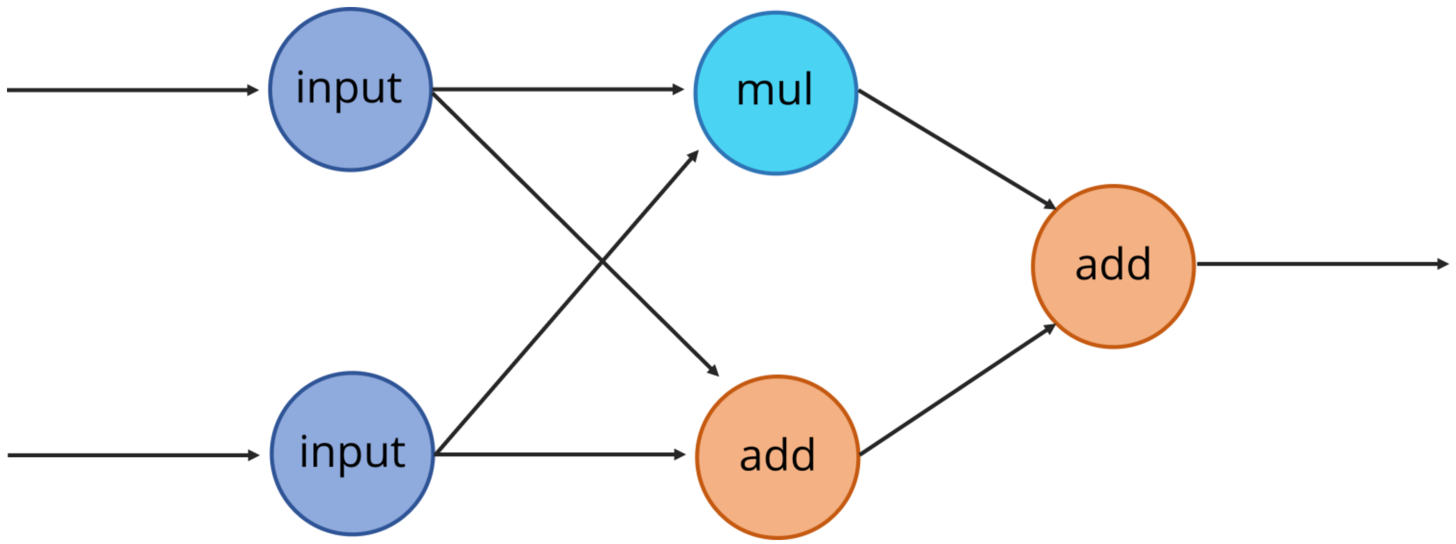
Nodes represent mathematical operations.

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

TensorFlow: Dataflow Graph

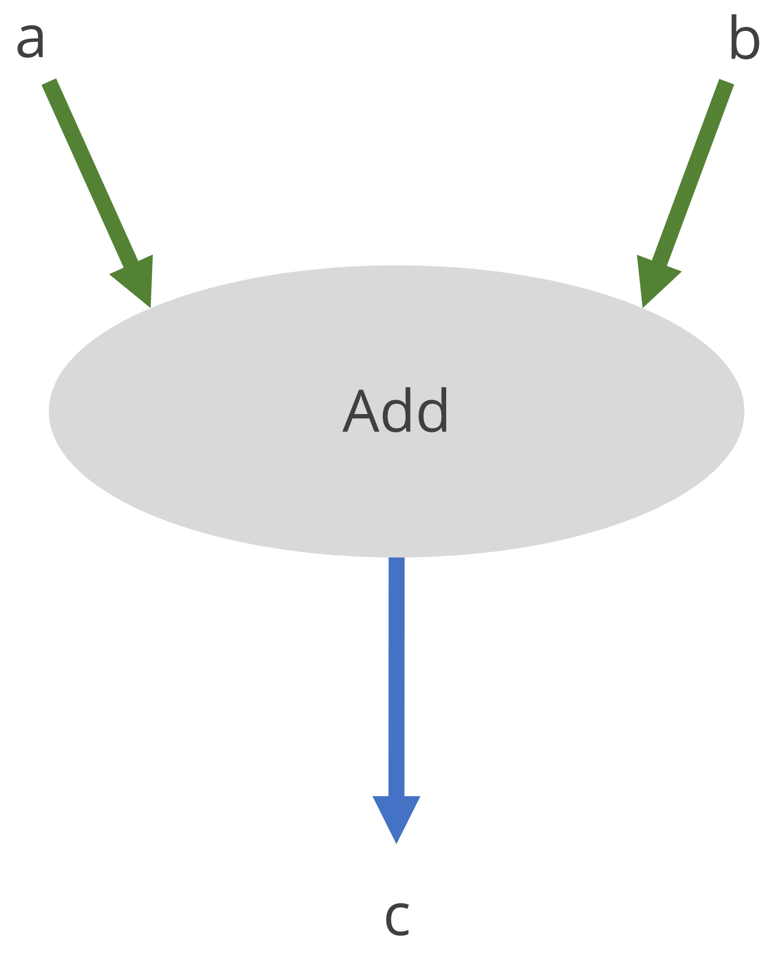
The following are the key points about TensorFlow

- TensorFlow uses a dataflow graph to represent your computation.
- Dataflow is a common programming model for parallel computing.
- TensorFlow optimizes graph execution by rearranging operations and leveraging parallelism for efficient computation.



How Does TensorFlow Work?

In the following image, Add is a node that represents the addition operation and a and b are the input tensors producing the output tensor c.



The flexible nature of this architecture allows the use of an API.

TensorFlow provides several APIs to deploy computations to one or more Central Processing Units (CPUs) or Graphics Processing Units (GPUs) in a desktop server or mobile device.

Benefits of Using Graphs

The benefits of using graphs include:

Parallelism	It is easy for the system to identify operations that can be executed parallelly.
Distributed execution	It is possible for TensorFlow to partition programs across multiple devices, CPUs, GPUs.
Compilation	It helps to generate faster code.
Portability	It can build a dataflow graph in Python, store it in a saved model, and restore it in a C++ program.

What Are the Categories of TensorFlow APIs?

TensorFlow APIs can be divided into two broad groups:

Low-level API	High-level API
Example: TensorFlow Core	Example: tf.contrib.learn
It is recommended for deep learning researchers.	It is an API that is built upon another API called the TensorFlow Core.
It provides finer levels of control over the models.	It is easier to learn and is more user-friendly than TensorFlow Core.
	It allows for repetitive chores to become easier and more consistent between various users.

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.

Importance of TensorFlow

There are various reasons why users should care about TensorFlow:

1. Provides abstraction through a ready library
2. Offers free graph visualization
3. Enables greater developer control through its flexible architecture
4. Makes Google's innovations accessible to all

Importance of TensorFlow

There are various reasons why users should care about TensorFlow:

5. Supports for multiple devices
6. Promotes artificial intelligence (AI) development
7. Is compatible with (GPUs)
8. Leverages parallelism to reduce memory use

Advantages of TensorFlow

The advantages for TensorFlow are:

- Flexibility
- Parallel computation
- Multiple environment friendly
- Largecommunities
- High-level abstraction

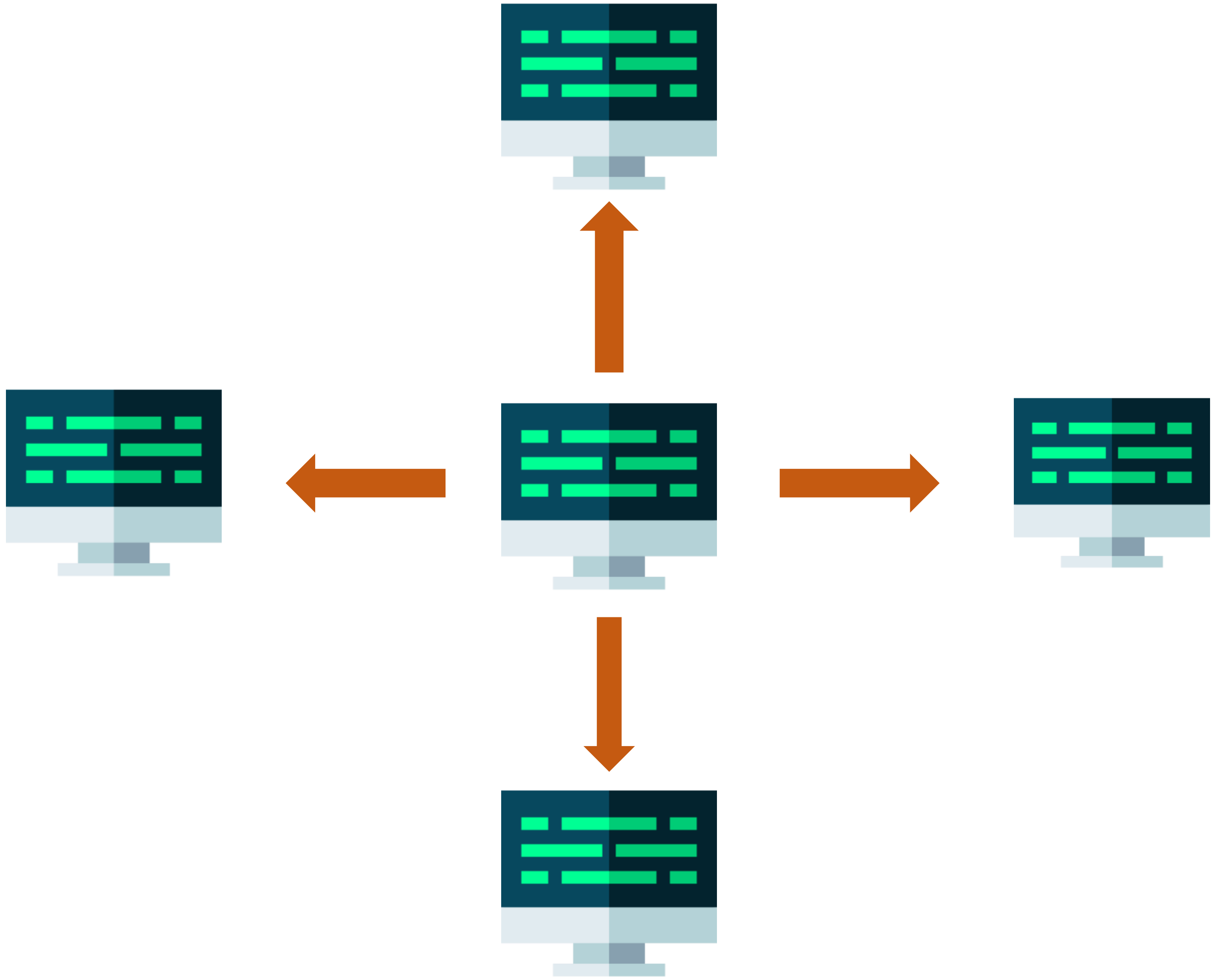
TensorFlow: Flexibility

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

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

Runs on desktop and mobile devices such as:

- Linux
- macOS
- iOS
- Android
- Raspberry Pi
- Windows

TensorFlow: Large Community

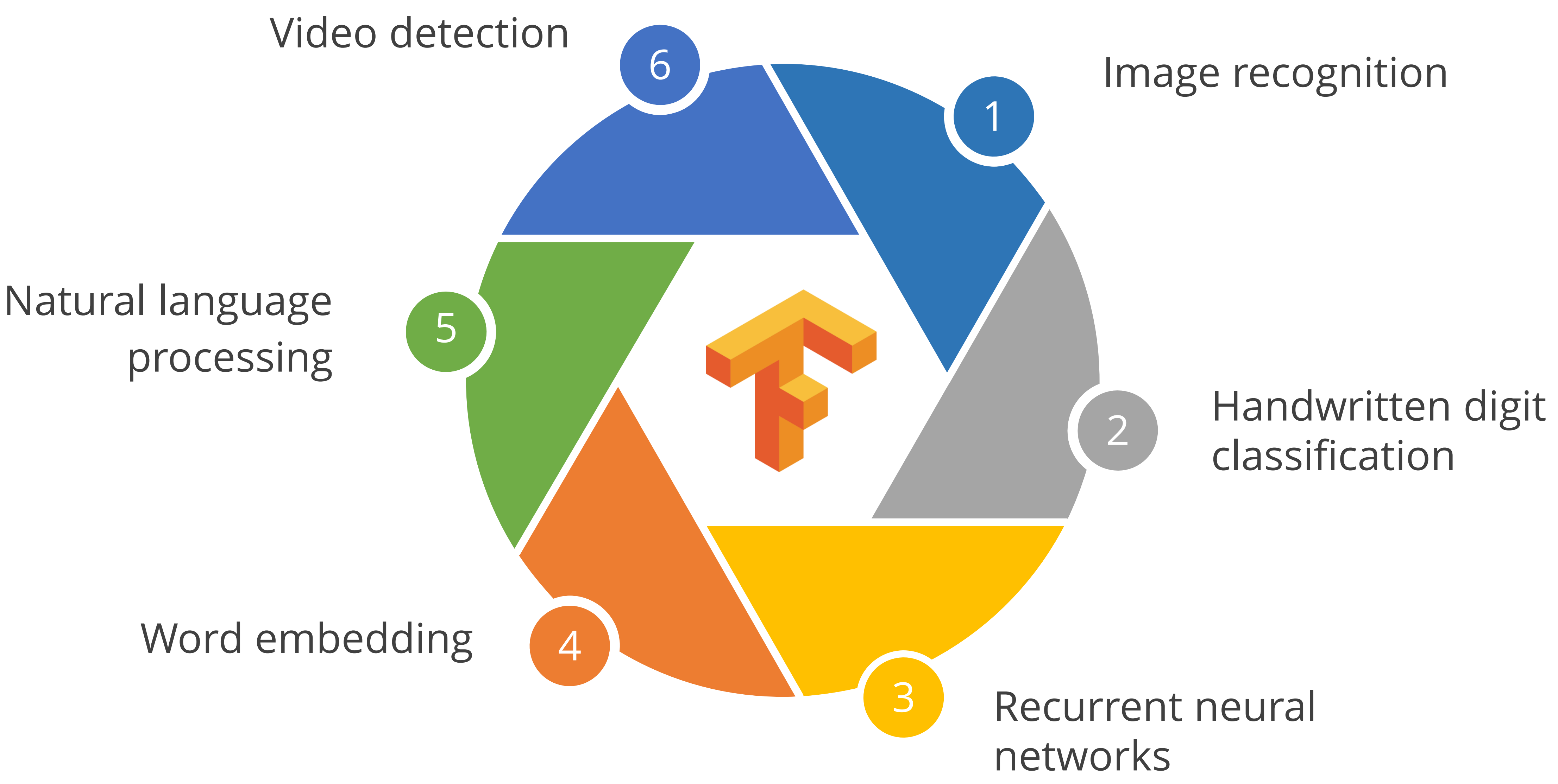
- Is one of the most popular open-source projects on GitHub
- Has a dedicated team of passionate and helpful developers
- Has a growing community contributing to improve it

TensorFlow: High-Level Abstraction

The benefits of high-level abstraction in TensorFlow are:

Application of TensorFlow

TensorFlow can train and run deep neural networks for:



Companies Using TensorFlow

Some of the top companies that use TensorFlow are:

Uber

SAP

DeepMind

Coca-Cola

Dropbox

Companies Using TensorFlow

Some of the top companies that use TensorFlow are:

eBay

Intel

Qualcomm

Twitter

Google

Discussion: TensorFlow

Duration: 10 minutes

- What is TensorFlow?
  Answer: TensorFlow is an open-source machine learning framework developed by Google.

- What are the features of TensorFlow?
  Answer: TensorFlow offers features like graph-based computation, automatic differentiation, flexibility, scalability, high-level APIs, and production deployment capabilities.

Assisted Practices

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

- 5.01_Introduction to Tensors
- 5.04_Training DNN Using TensorFlow

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

ASSISTED PRACTICE

Installation of TensorFlow

Prerequisites for 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, which 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

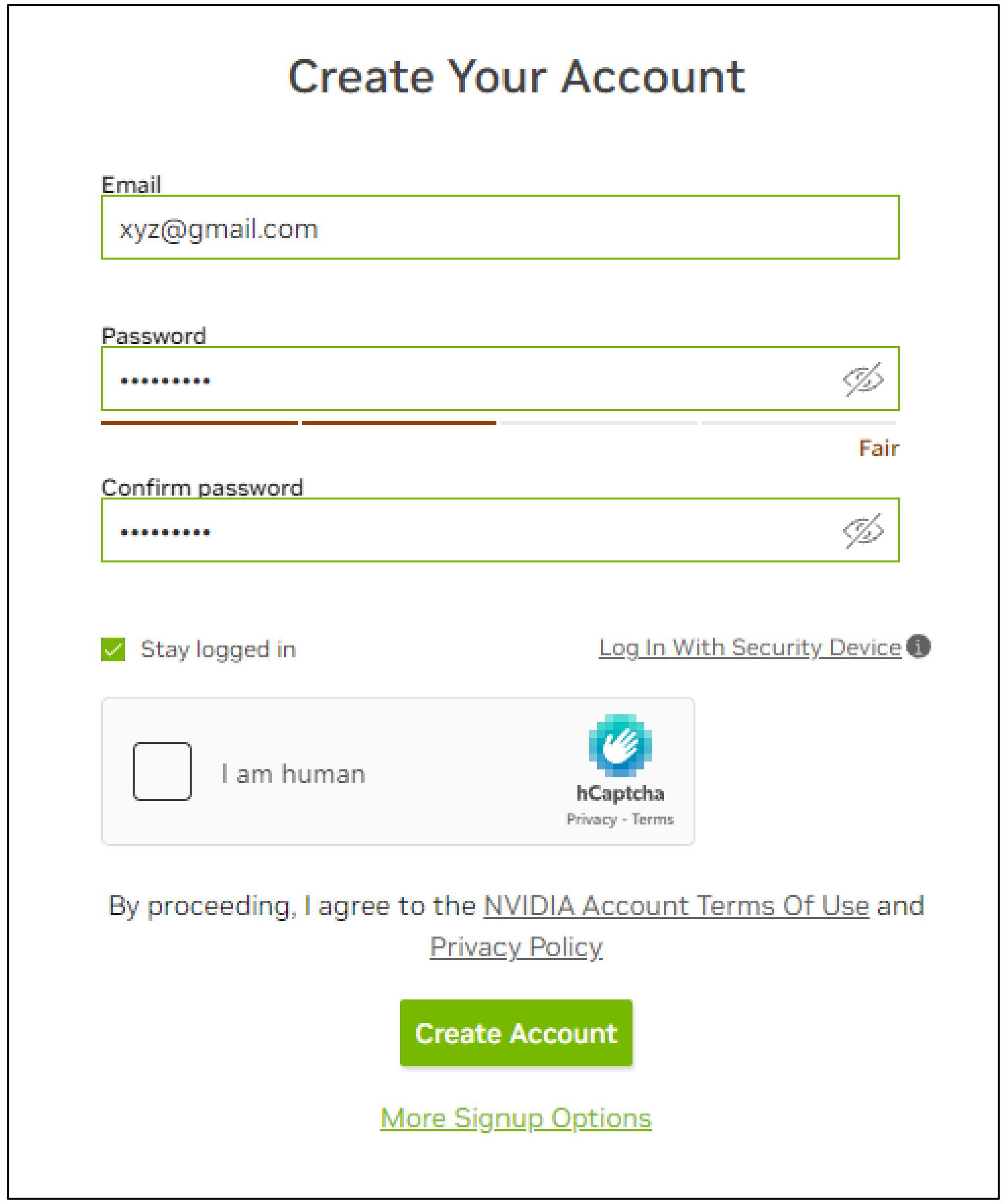
NVIDIA CUDA

Download and install the right version of the NVIDIA CUDA toolkit.

For 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



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

Download the cuDNN library.

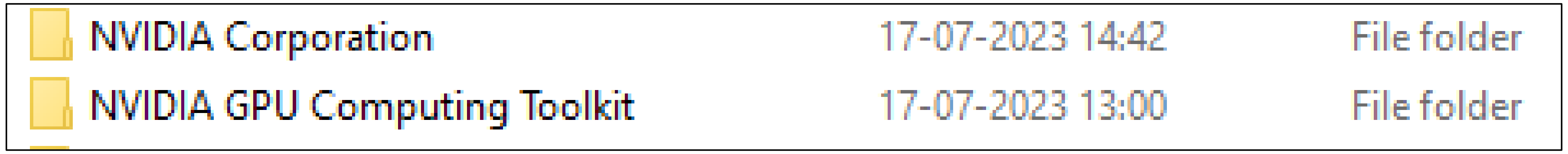
Setting Up the System

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**



Setting Up the System

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

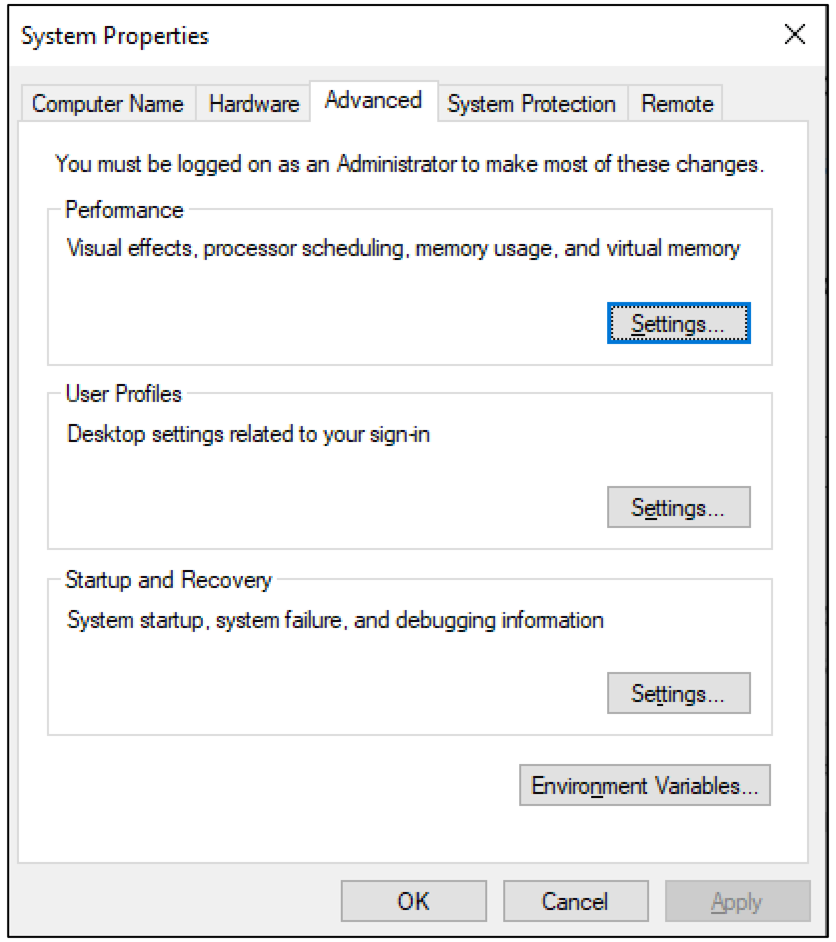
s PC › OS (C:) › Program Files › NVIDIA GPU Computing Toolkit › CUDA › v11.2

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

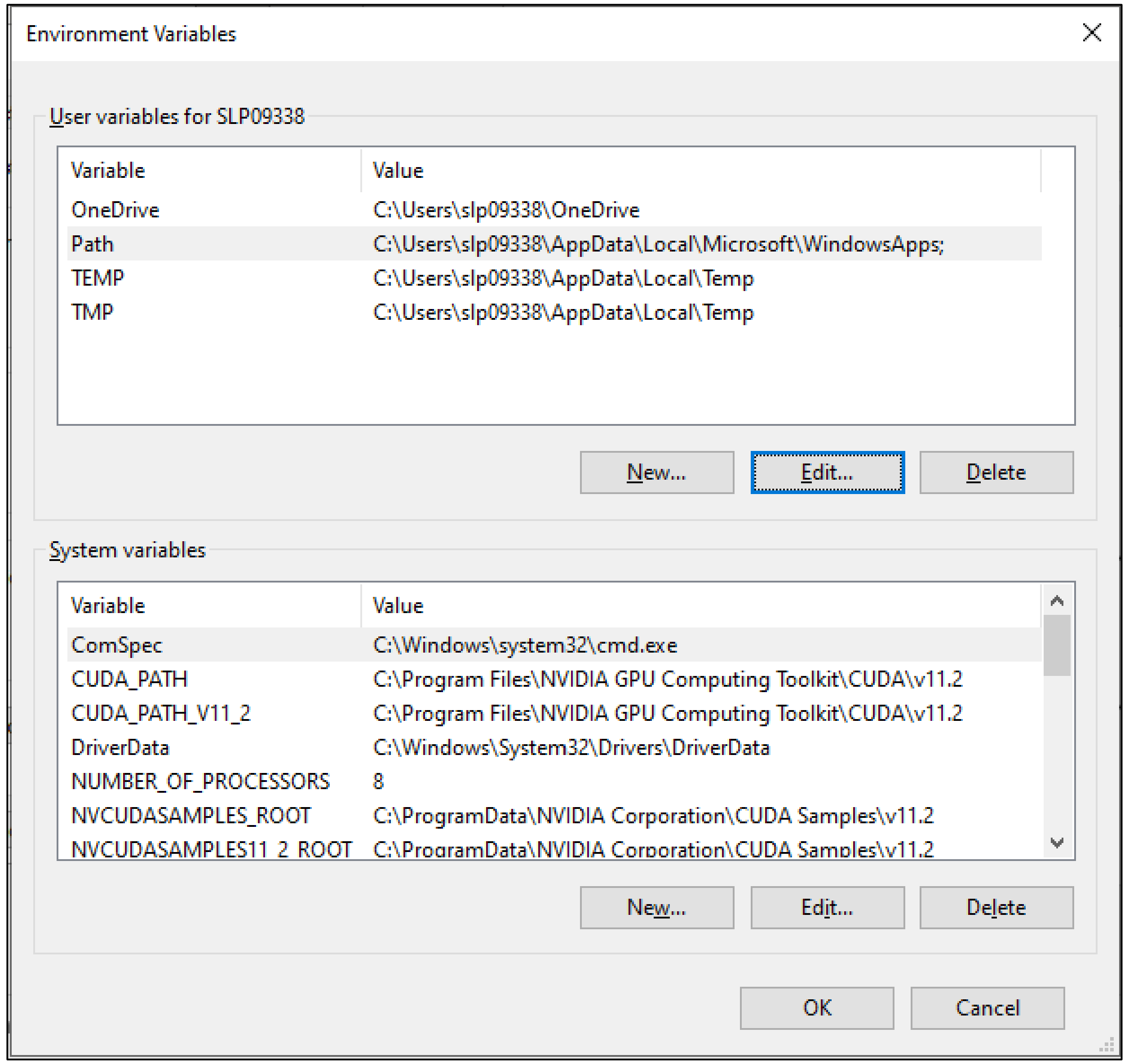


Next, open **System Properties** and go to **Environment Variables**.

Setting Up the System

8

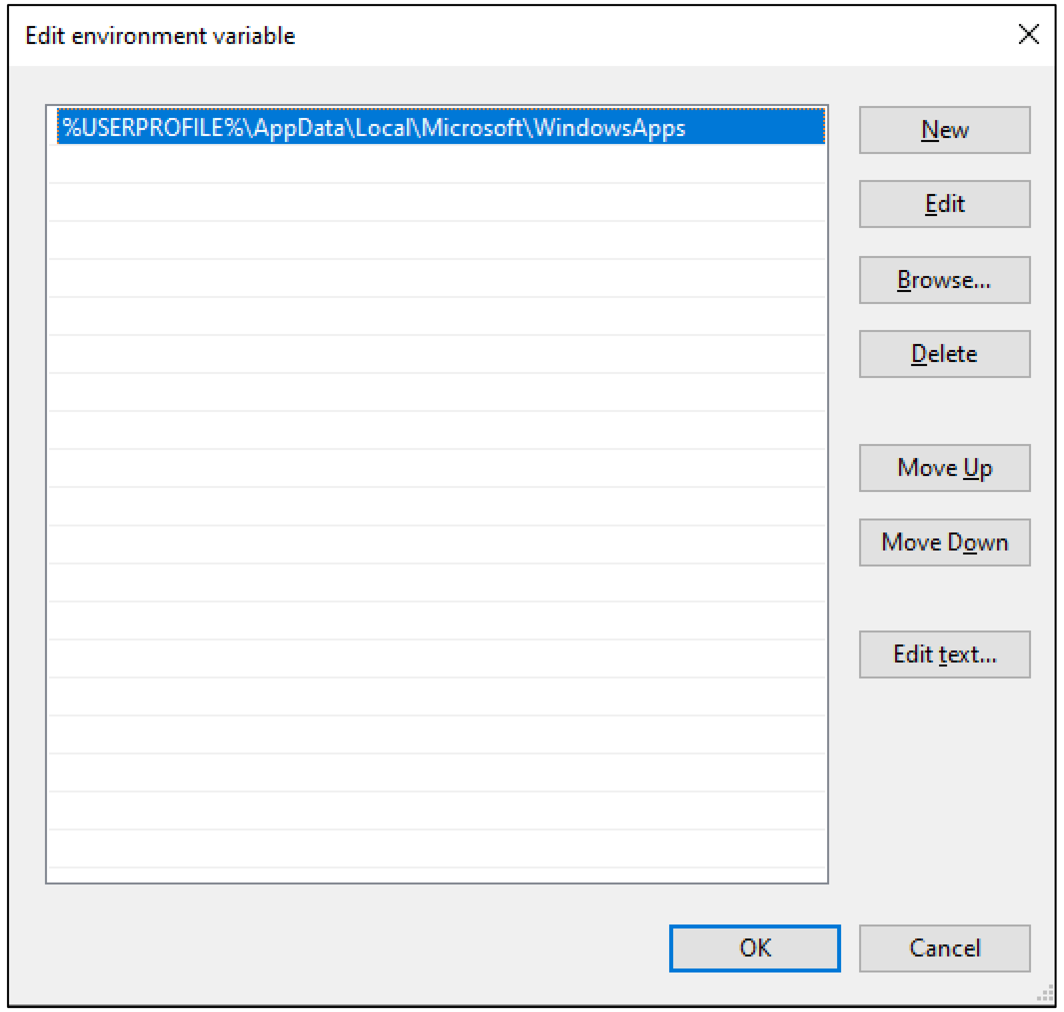
Double-click on the **Path** here



Setting Up the System

9

Click on **New**

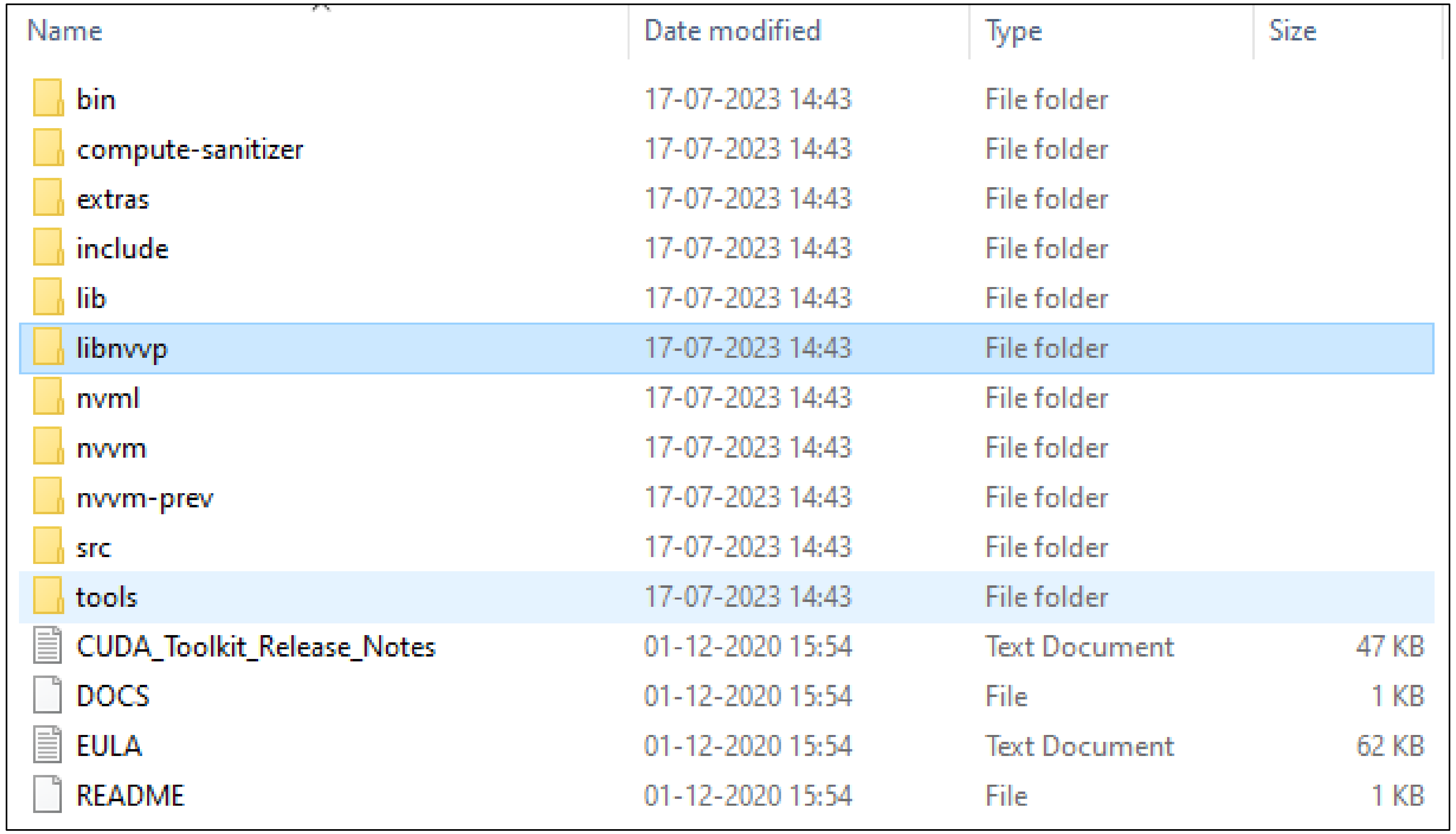


Paste the copied path or location at the bottom and press **OK**.

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:

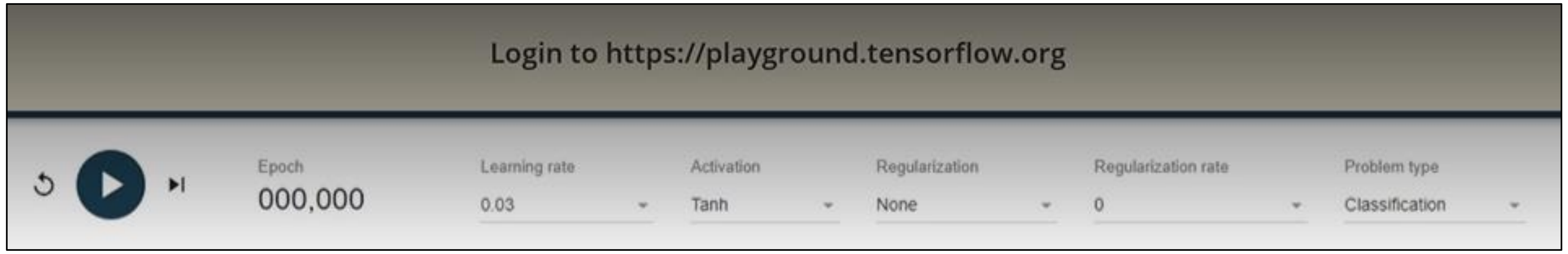
```
[1]: import tensorflow as tf
     print(tf.__version__)

2.8.0
```

TensorFlow Playground

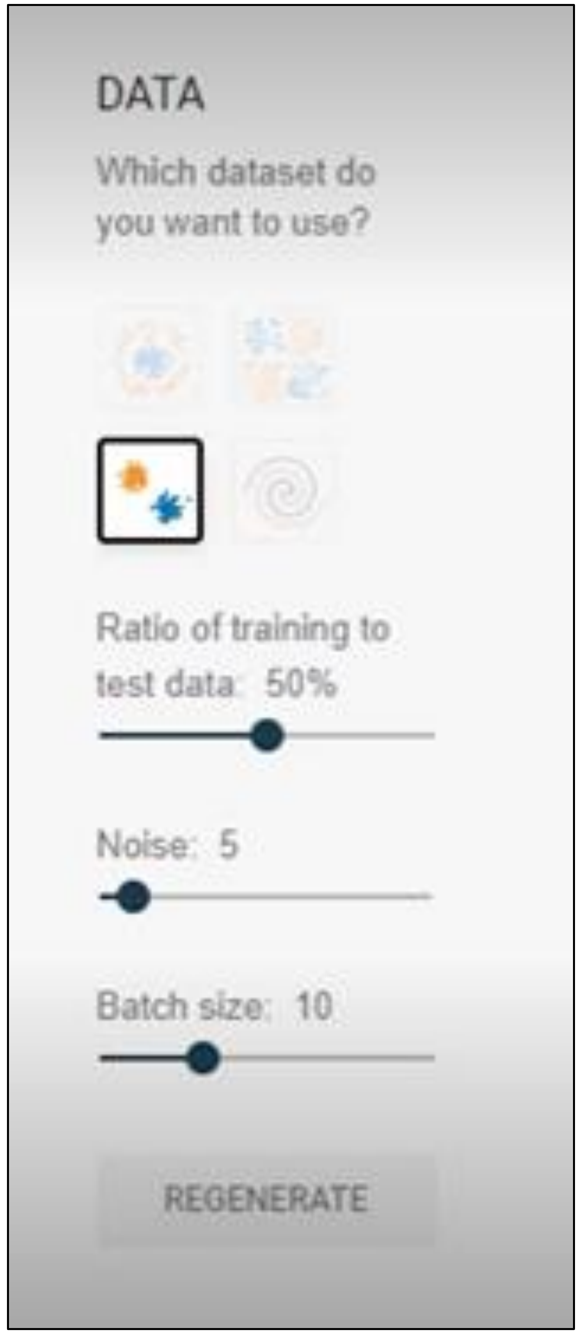
Hands-on with the TensorFlow Playground

Step 1: Log into the website



Example: Simple Classification Problem

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

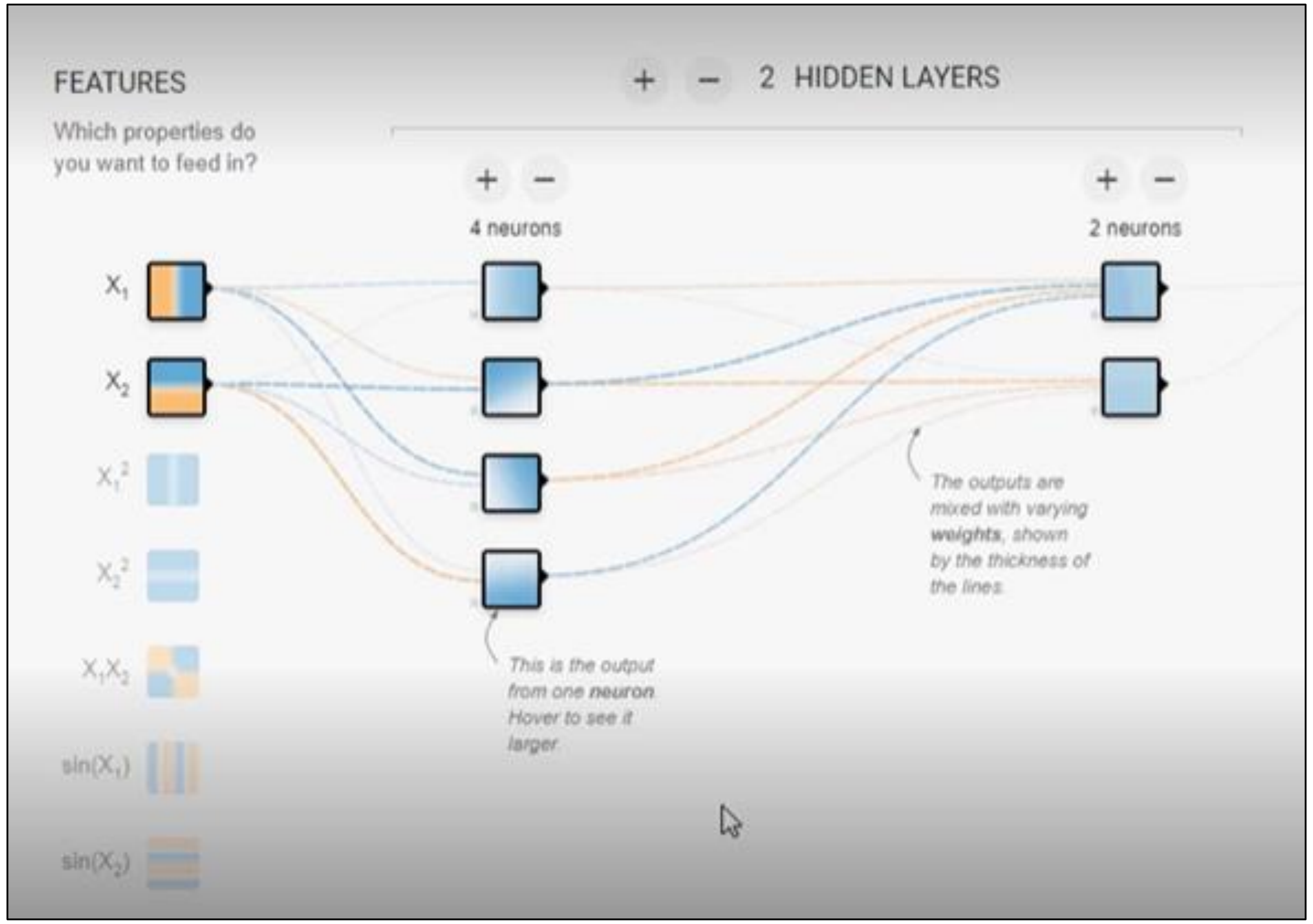


Example: Simple Classification Problem

Step 3: Set the features for the epoch, learning rate, activation, regularization, regularization rate, and problem type

Example: Simple Classification Problem

Step 4: Set the fully connected dense layer

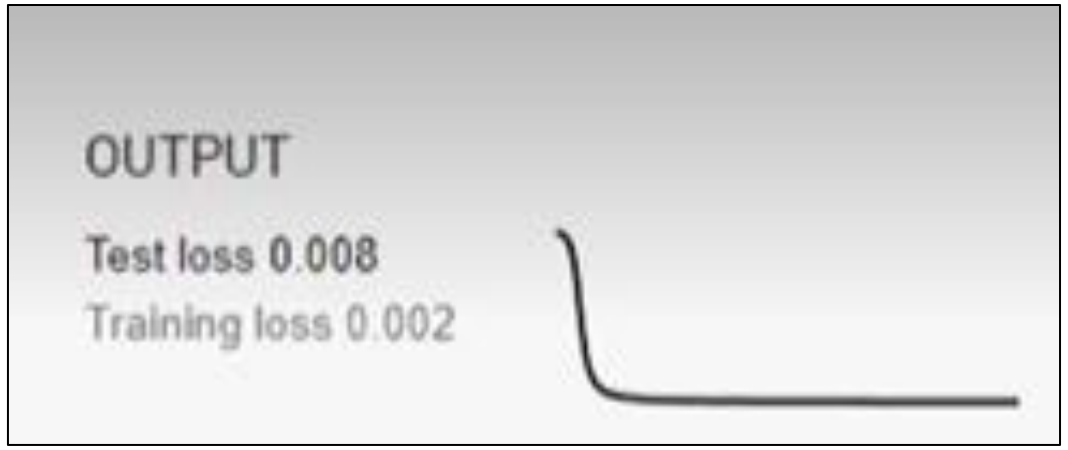


Example: Simple Classification Problem

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

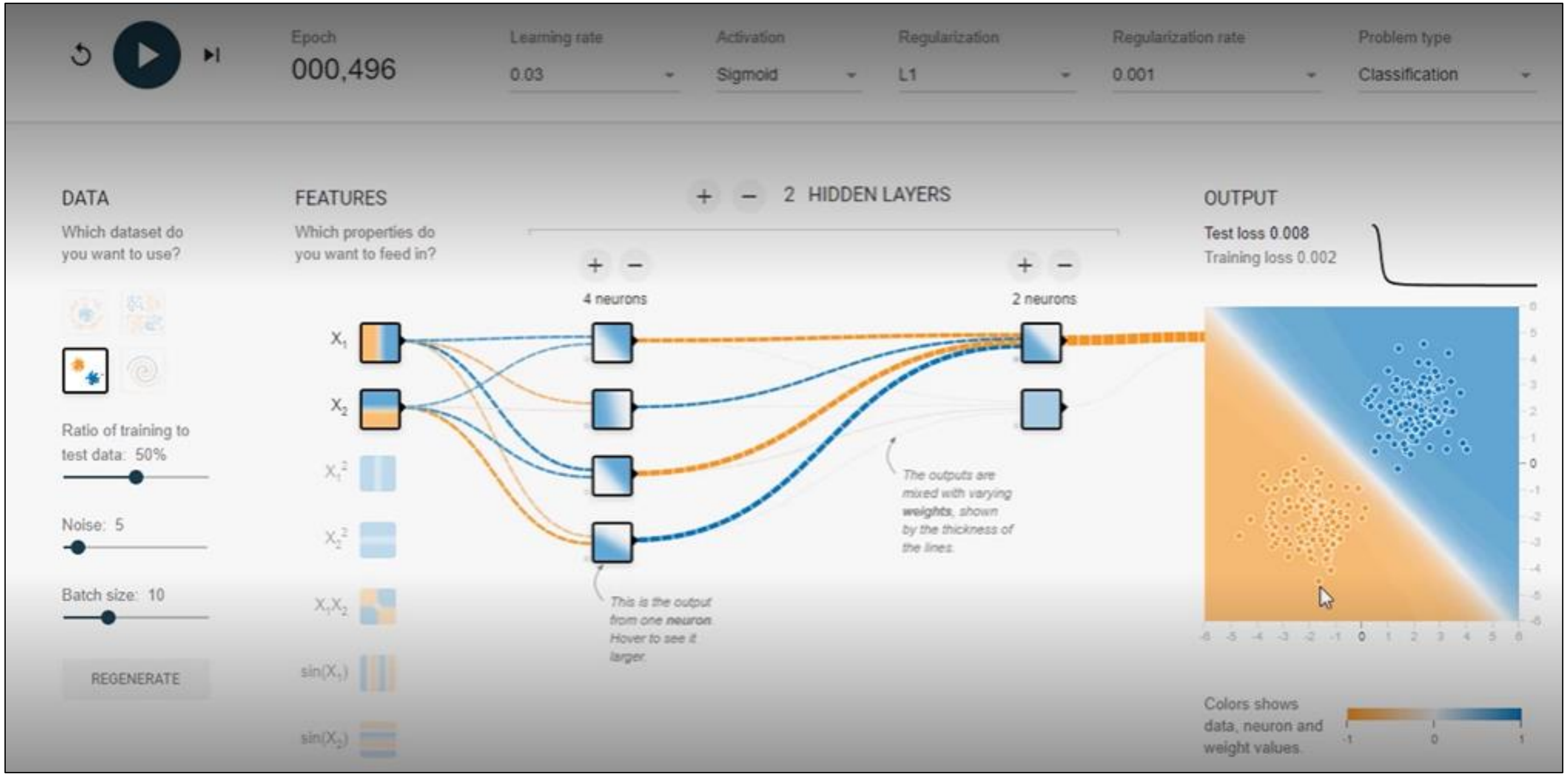
Example: Simple Classification Problem

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



Example: Simple Classification Problem

Result: As seen, within the 496 epoch, a decision boundary is built between two classes**.**



Assisted Practices

Let’s understand the concept of sequential and functional API in TensorFlow using Jupyter Notebooks.

- 5.05_Sequential_APIs_in_TensorFlow
- 5.06_Functional_APIs_in_TensorFlow

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

ASSISTED PRACTICE

TFLearn

Discussion

Discussion: TFLearn

Duration: 10 minutes

- What is TFLearn?
- What are the features of TFLearn?

What Is TFLearn?

TFlearn is a modular and transparent deep-learning library built on top of TensorFlow. It was designed to provide a higher-level API to TensorFlow to facilitate and speed up experiments while remaining fully transparent and compatible with it.

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, 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, large-scale projects; advanced users	Quick experimentation; beginners and intermediate users
Integration and visualization	Integrates with TensorBoard and other advanced tools	Integrates with TensorFlow; simpler without advanced tools

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 using multiple CPUs or GPUs

Installation of TFLearn

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

For the latest stable version:

- pip install tflearn

Workflow of the TFLearn Model

The simple method in TFLearn appears to be the following:

1. Create an input layer first.
2. Pass the input object to create further layers.
3. Add the output layer.
4. Create the neural network model using an estimator layer such as regression.
5. Create a model from the net created in the previous step.
6. Train the model with the model.fit() method.
7. Use the trained model to predict or evaluate.

Layers of TFLearn

Currently available layers of TFLearn are:

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

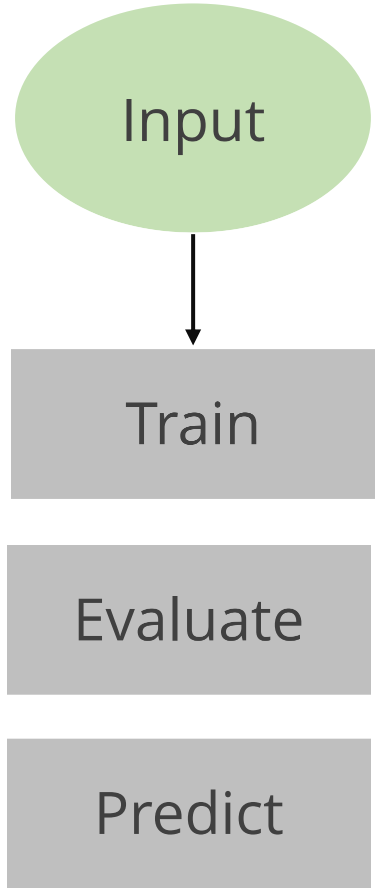
Here is a list of all the currently available operations:

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



- Any TensorFlow graph can be trained using the assistance functions offered by TFLearn.
- By introducing real-time monitoring, batch sampling, moving averages, TensorBoard logs, data input, ‘etc’. it is appropriate to make training more convenient.
- 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's how it's 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 TrainOp 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)
```

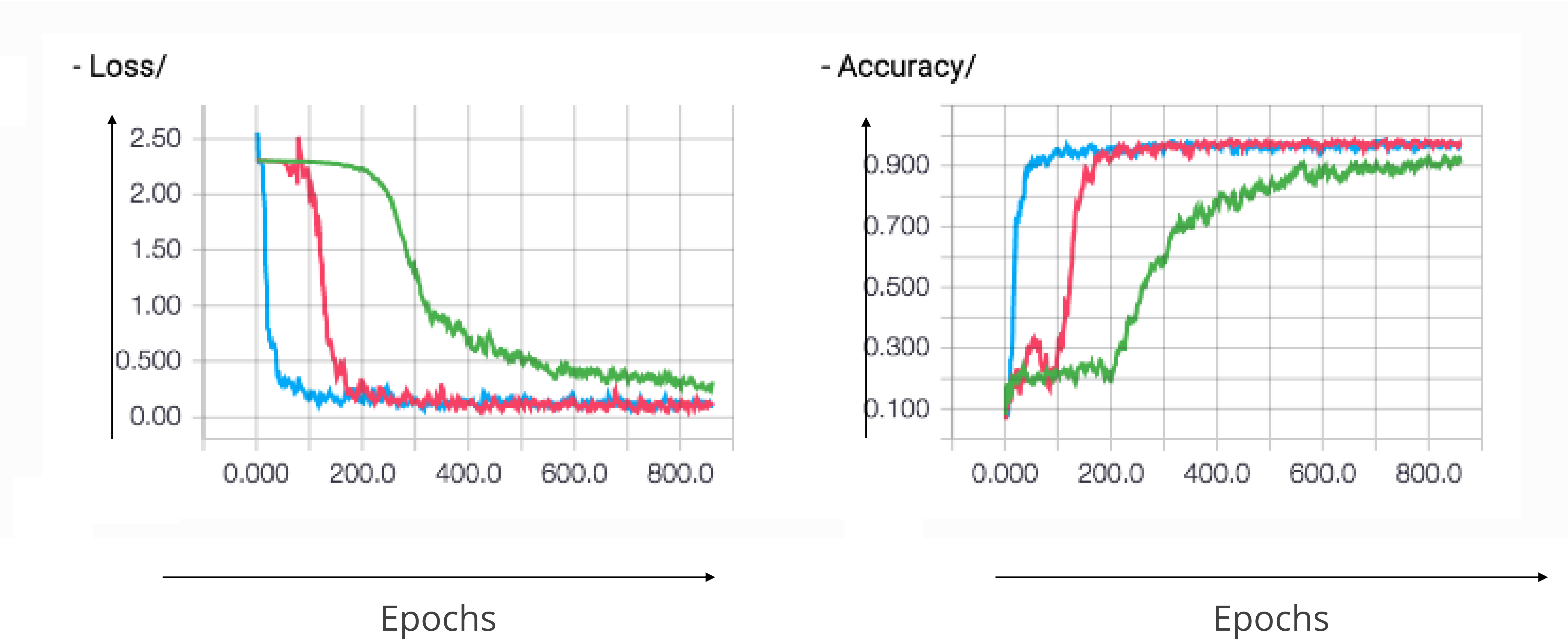
Visualization

TFLearn can manage a lot of useful logs. Currently, TFLearn supports a verbose level to automatically manage summaries:

- 1: Loss and Metric (Best Speed)
- 2: Loss, Metric, and Gradients
- 3: Loss, Metric, Gradients, and Weights
- 4: Loss, Metric, Gradients, Weights, Activations, and Sparsity (Best Visualization)

Visualization: Loss and Accuracy

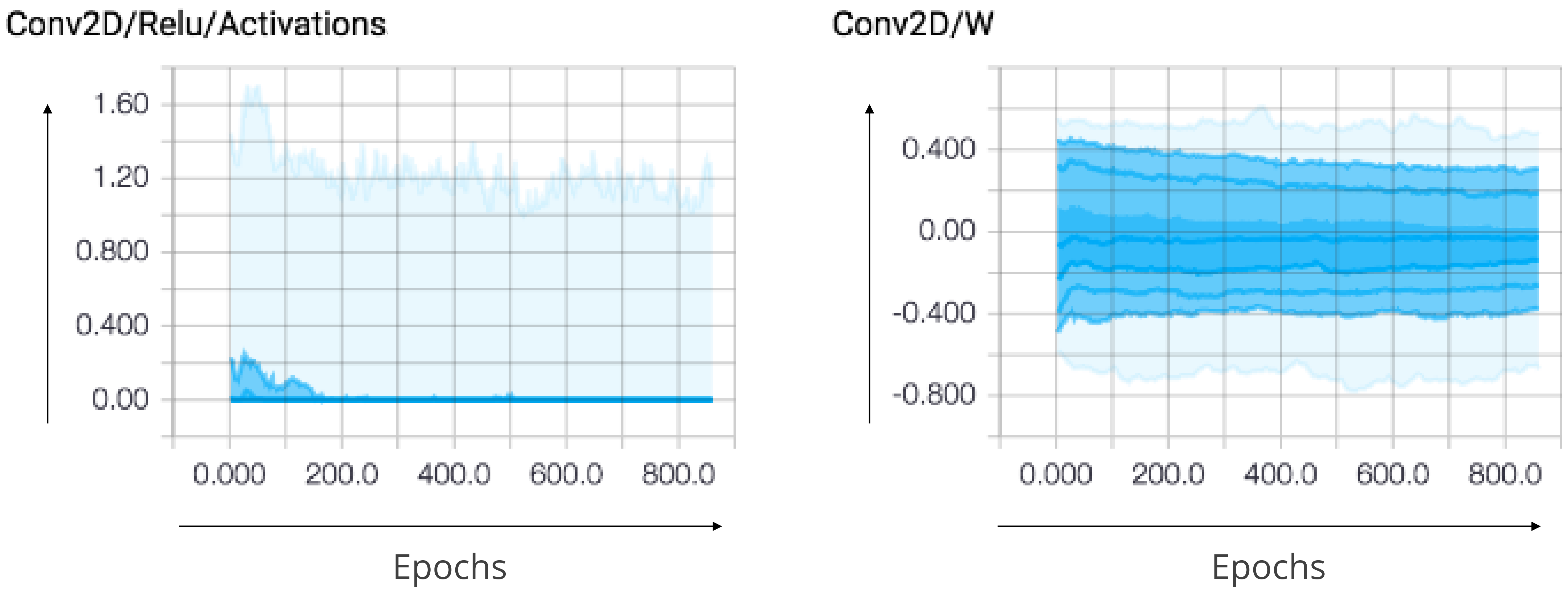
Visualizing loss and accuracy aids in analyzing and optimizing neural network training.



http://tflearn.org/img/loss_acc.png

Visualization: Layers

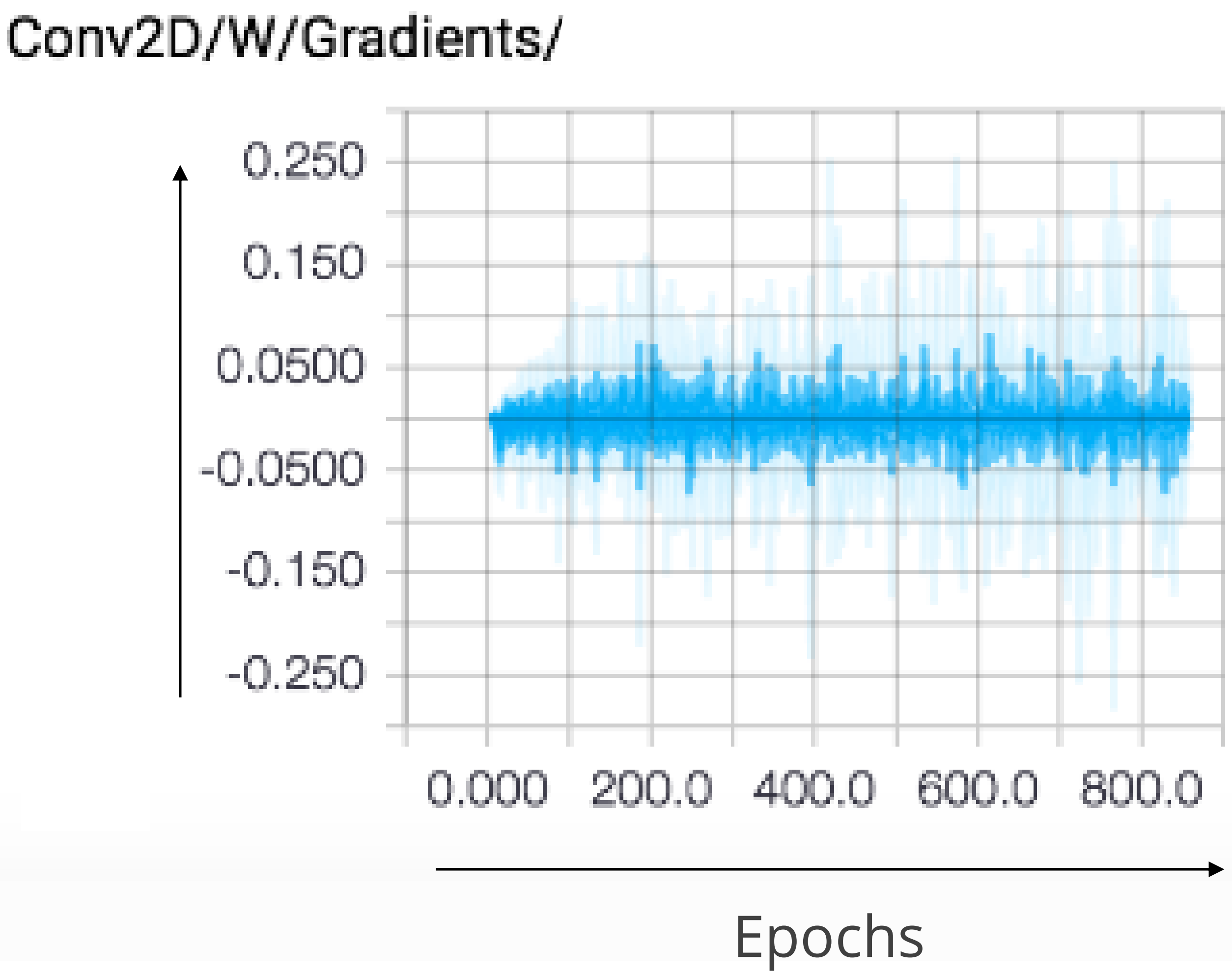
Visualizing the layers between convolutional operations and convolutional weight layers provides insights into feature extraction and transformations within a convolutional neural network.

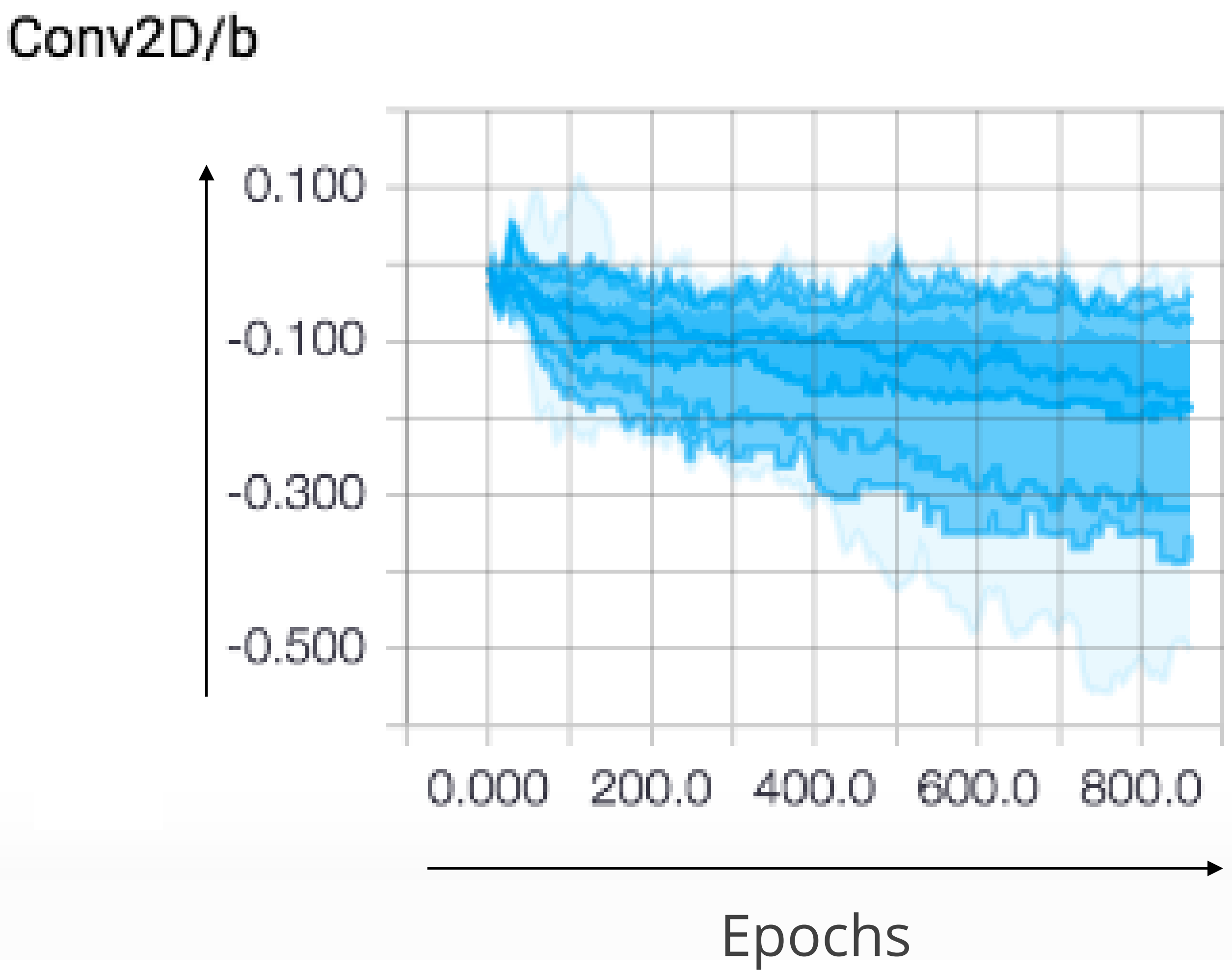


http://tflearn.org/img/layer_visualization.png

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.





http://tflearn.org/img/layer_visualization.png

Discussion: TFLearn

Duration: 10 minutes

- What is TFLearn?
  Answer: TFLearn is a high-level deep learning library that is built on top of TensorFlow.

- What are the features of TFLearn?
  Answer: TFLearn offers easy-to-use APIs, high-level abstractions, predefined neural network layers, visualization tools, and model-saving capabilities.

Example for Weights Persistence

Weight Persistence

It is the process of saving and loading learned parameter values (weights) of a trained model for future use or deployment.

The following are the examples for maintaining a model's weights:

Example using TensorFlow:

```
#Save the weights
model.save_weights('my_model_weights.npy')

#Load a model
torch.save(model.state_dict(), 'my_model_weights.pth')
```

Weight Persistence

A layer variable can be directly retrieved by using the layer name, or indirectly by using the **W** or **b** attributes that are associated with the layer's returned tensor.

Syntax:-

```
# Let's create a layer
fc1 = fully_connected(input_layer, 64, name="fc_layer_1")

# Using Tensor attributes (Layer will supercharge the returned Tensor with
weights attributes)
fc1_weights_var = fc1.W
fc1_biases_var = fc1.b

# Using Tensor name
fc1_vars = tflearn.get_layer_variables_by_name("fc_layer_1")
fc1_weights_var = fc1_vars[0]
fc1_biases_var = fc1_vars[1]
```

Weight Persistence

The TFLearn model classes implement the **get_weights** and **set_weights** methods to obtain or set the values of these variables:

Syntax:-

```
input_data = tflearn.input_data(shape=[None, 784])
fc1 = tflearn.fully_connected(input_data, 64)
fc2 = tflearn.fully_connected(fc1, 10, activation='softmax')
net = tflearn.regression(fc2)
model = DNN(net)

# Get weights values of fc2
model.get_weights(fc2.W)

# Assign new random weights to fc2
model.set_weights(fc2.W, numpy.random.rand(64, 10))
```

Example for Weights Persistence

The following is an illustration of how to save, restore, and retrieve weights for models:

Example:-

```
pip install tensorflow
pip install tflearn

from __future__ import absolute_import, division, print_function
import tflearn
import tflearn.datasets.mnist as mnist

# MNIST Data
x, y, testx, testy = mnist.load_data(one_hot=True)


# Model
input_layer = tflearn.input_data(shape=[None, 784], name='input')
dense1 = tflearn.fully_connected(input_layer, 128, name='dense1')
dense2 = tflearn.fully_connected(dense1, 256, name='dense2')
softmax = tflearn.fully_connected(dense2, 10, activation='softmax')
regression = tflearn.regression(softmax, optimizer='adam',
learning_rate=0.001,
loss='categorical_crossentropy')
```

Example for Weights Persistence

A classifier model with a model checkpoint that automatically saves the model and weights for later use or evaluation during training

Example:-

```
# Define classifier, with model checkpoint (autosave)
model = tflearn.DNN(regression, checkpoint_path='model.tfl.ckpt')

# Train model, with model checkpoint every epoch and every 200 training steps.
model.fit(x, y, n_epoch=1,
validation_set=(testx, testy),
show_metric=True,
snapshot_epoch=True, # Snapshot (save & evaluate) model every epoch.
snapshot_step=500, # Snapshot (save & evalaute) model every 500 steps.
run_id='model_and_weights')
```

Output:

```
Training Step: 859  | total loss: 0.34583 | time: 9.720s
| Adam | epoch: 001 | loss: 0.34583 - acc: 0.9073 -- iter: 54976/55000
Training Step: 860  | total loss: 0.34946 | time: 11.070s
| Adam | epoch: 001 | loss: 0.34946 - acc: 0.9040 | val_loss: 0.30898 - val_acc: 0.9104 -- iter: 55000/55000
--
```

Example for Weights Persistence

The following is an illustration of how to save and load the model:

Example:-

```
# Manually save model
model.save("model.tfl")

# Load a model
model.load("model.tfl")
```

Example for Weights Persistence

The following is an illustration of how to retrieve and print the weights:

Example:-

```
# Retrieve a layer weights, by layer name:
dense1_vars =
tflearn.variables.get_layer_variables_by_name('dense1’)

# Get a variable's value, using model `get_weights`
method:
print("Dense1 layer weights:")
print(model.get_weights(dense1_vars[0]))

# Or using generic tflearn function:
print("Dense1 layer biases:")
with model.session.as_default():
print(tflearn.variables.get_value(dense1_vars[1]))
```

Output:

```
Dense1 layer weights:
[[-0.01585038 -0.02928603 -0.00553446 ...  0.01987983  0.01279698
  -0.00774803]
 [ 0.02107707  0.01029426  0.02392667 ... -0.0376506  -0.00430582
   0.00740492]
 [-0.02865655 -0.03132273 -0.02557028 ...  0.01762935  0.01508698
  -0.03276213]
 ...
 [-0.00673502  0.01680059 -0.00065119 ...  0.01528361  0.02910966
  -0.01013854]
 [-0.02275838  0.00490783 -0.00888035 ...  0.00838752  0.00470138
   0.00798579]
 [-0.01218346 -0.00784085  0.01447292 ... -0.00704181 -0.00181136
   0.00028389]]
Dense1 layer biases:
[-0.08959953  0.12752196 -0.03664213  0.04176922 -0.07898358  0.12317553
  0.0634919  -0.13710164  0.08044362  0.07381842  0.06128288  0.15354979
 -0.13706857 -0.05021407  0.0144396  -0.01752157  0.08866858 -0.18744537
  0.04111555 -0.14663197  0.0150964   0.10239395 -0.02510643  0.02693582
 -0.01874169 -0.07230382 -0.01998399 -0.04737619 -0.0342963  -0.19499879
```

Example for Weights Persistence

It is also possible to retrieve a layer's weights through its attributes **W** and **b**.

Example:-

```
# Get variable's value, using model `get_weights`
method:
print("Dense2 layer weights:")
print(model.get_weights(dense2.W))

# Or using generic tflearn function:
print("Dense2 layer biases:")
with model.session.as_default():
print(tflearn.variables.get_value(dense2.b))
```

Output:

```
Dense2 layer weights:
[[-0.02109558 -0.01622235 -0.00327621 ...  0.02710502 -0.0320264
   0.01232063]
 [-0.00848038  0.10373902 -0.00148136 ... -0.01688047  0.05441003
  -0.05912356]
 [-0.05946014 -0.02030221  0.01153812 ...  0.0587462  -0.00696223
   0.00998368]
 ...
 [ 0.00102621  0.00626076 -0.00897411 ...  0.02349314  0.01155679
  -0.04750484]
 [-0.06348789  0.02444146  0.02821664 ...  0.08660153  0.01717809
   0.01776999]
 [-0.05186261  0.0192213  -0.04345338 ...  0.0448567   0.04523651
   0.01415755]]
Dense2 layer biases:
[ 0.05906097 -0.02205005  0.1229835   0.09705604  0.05766064  0.13980703
 -0.00093629  0.02585382 -0.01201913  0.05737786  0.01203733  0.07003123
 -0.00583897 -0.00446162 -0.02012413 -0.05532632  0.15479377  0.03251151
  0.08060881 -0.13036777 -0.03706704  0.02625279  0.05243282  0.10330328
  0.0557044  -0.11237332  0.03649668 -0.06363929 -0.08305153  0.10508173
```

Introduction to Keras

Discussion

Discussion: Keras

Duration: 10 minutes

- What is Keras?
- What are the features of Keras?
- Why is Keras used?

Keras

Keras is a high-level neural network library written in Python that can run TensorFlow. It is:

- Fast
- Easy to implement
- Modular in nature

It was developed by Francois Chollet, a Google AI engineer.

What Is Keras?

A high-level neural network API, written in Python

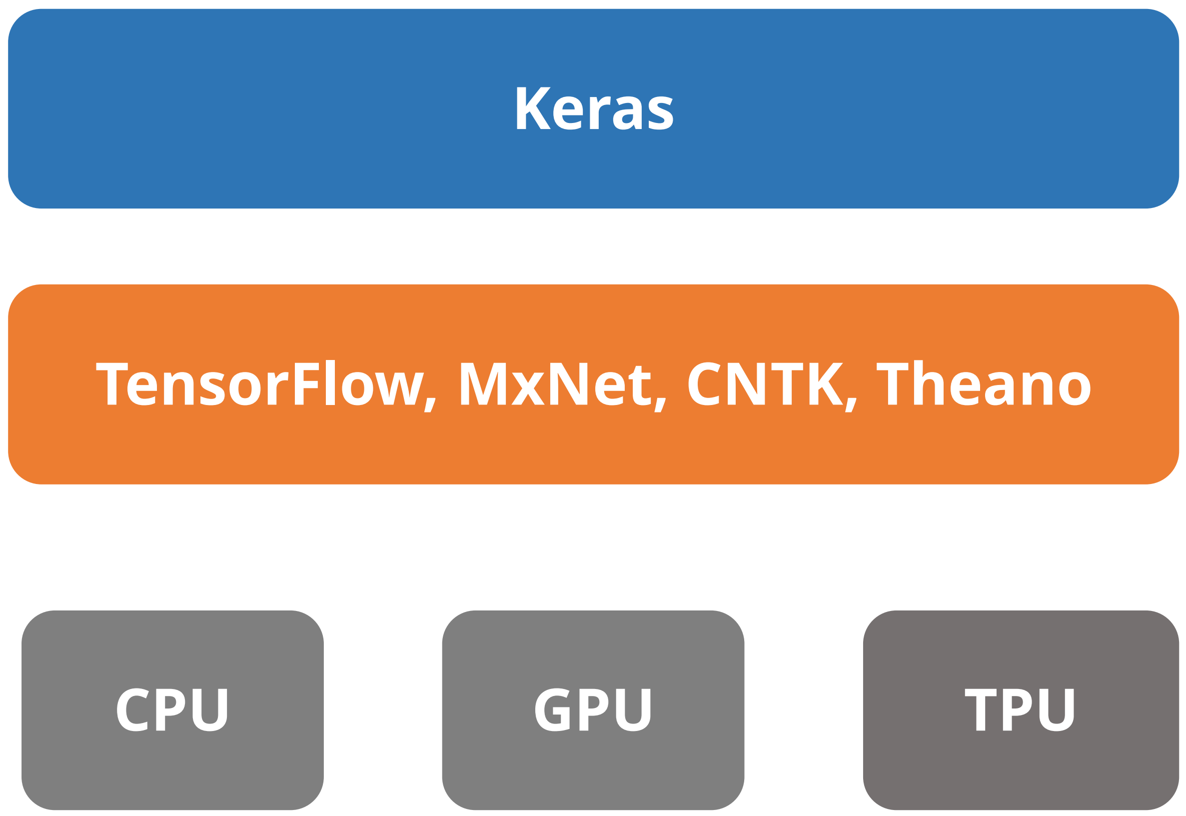
Most powerful and easy to use for developing and evaluating deep learning models

Keras

Runs seamlessly on CPU and GPU

Keras: Backends

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

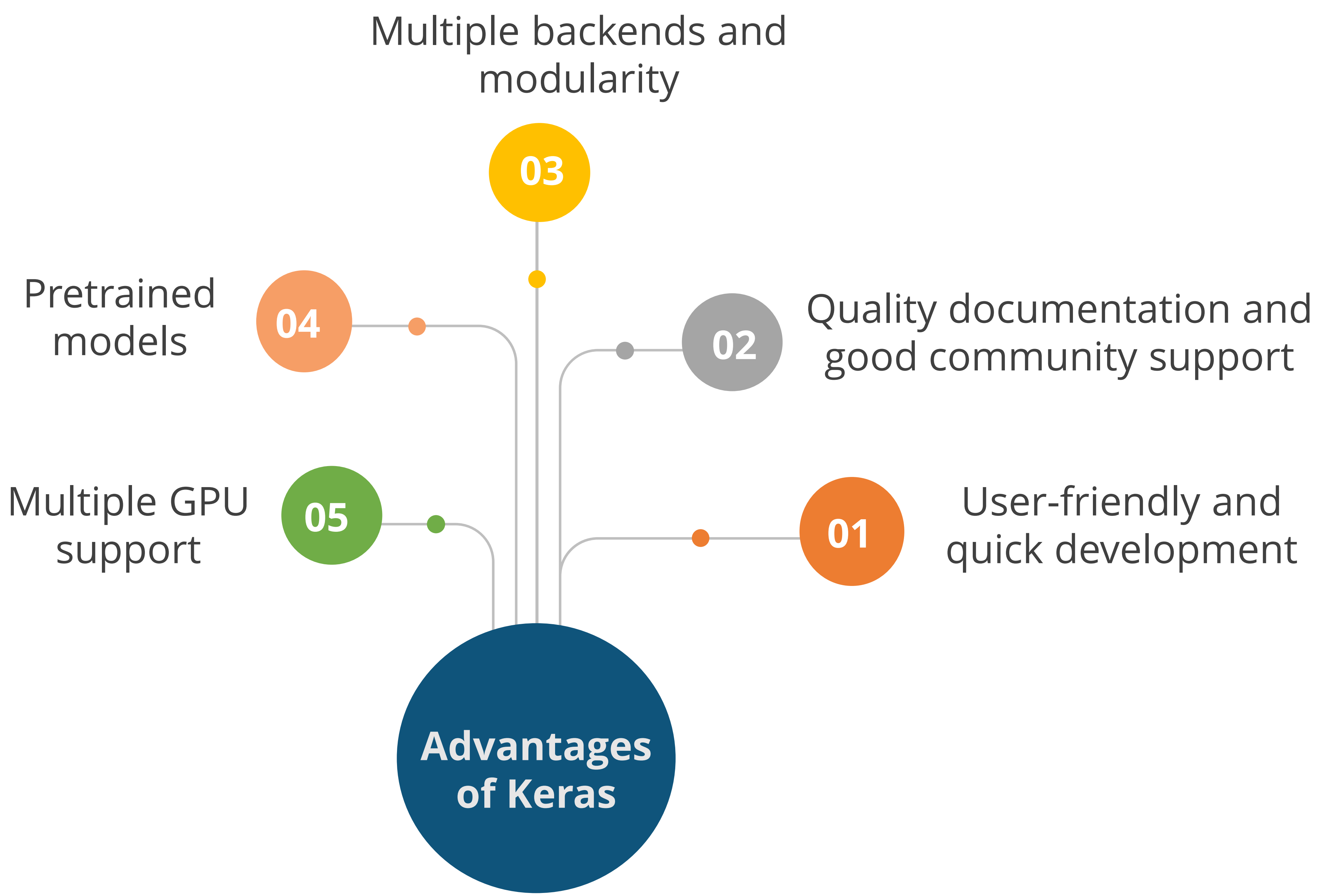


Why use Keras?

- Allows easy and fast prototyping
- Supports both convolutional networks, recurrent networks, and a combination of both
- Provides clear and actionable feedback for user error
- Follows best practices for reducing cognitive load

Advantages of Keras

Keras has the following advantages:



Advantages of Keras

User-friendly and quick development

Keras has a user-friendly API, and it is very easy to create neural network models with it. It is also good for implementing algorithms and natural language processing.

Advantages of Keras

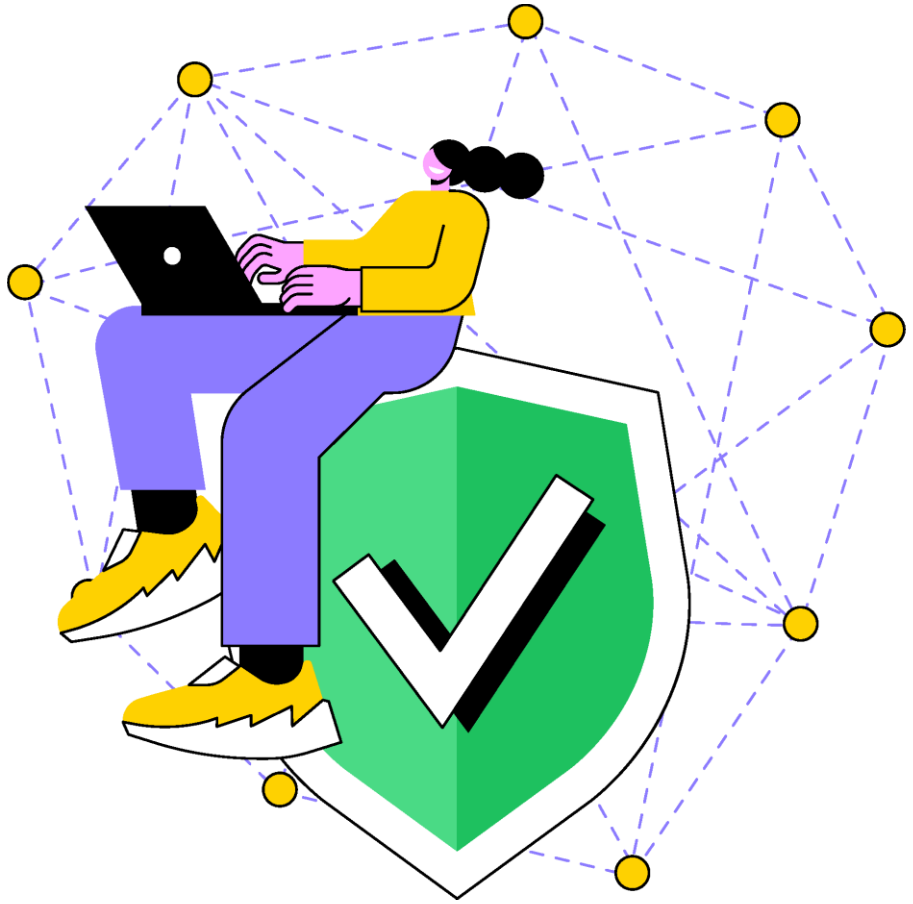
Quality documentation and good community support

Keras has a very good documentation process and introduces an organized and sequential method for each function. Keras also has great community support with several open-source platforms with community codes.

Advantages of Keras

Multiple backends and modularity

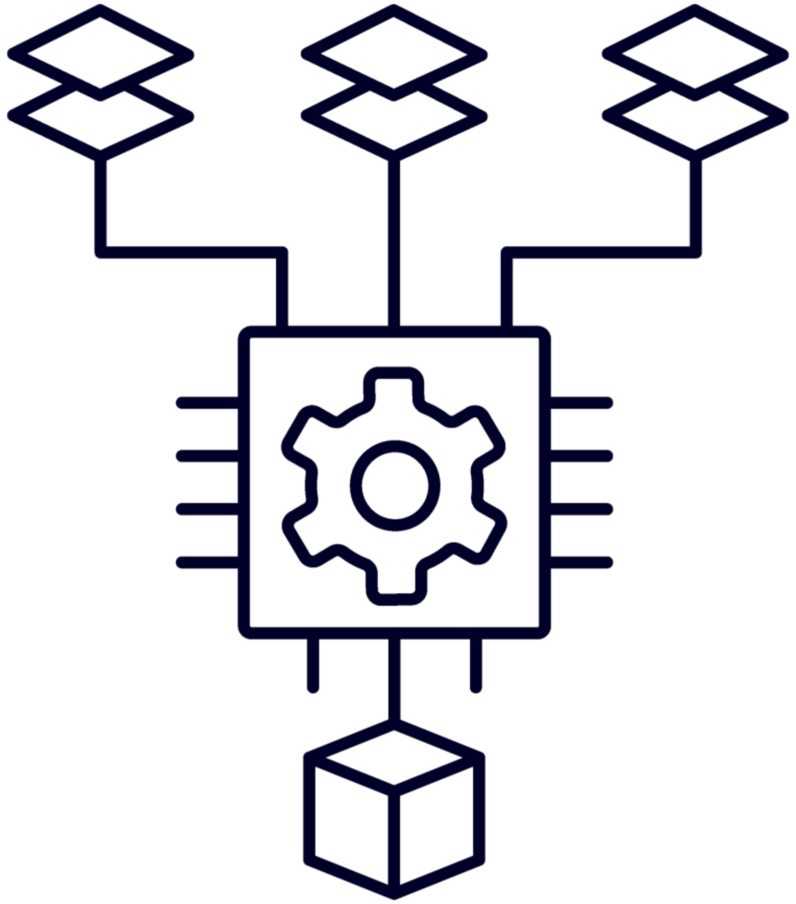
One can train a Keras model on one backend and test its results on another. Here, the developer must write the name of the backend in the configuration file.



Advantages of Keras

Pretrained models

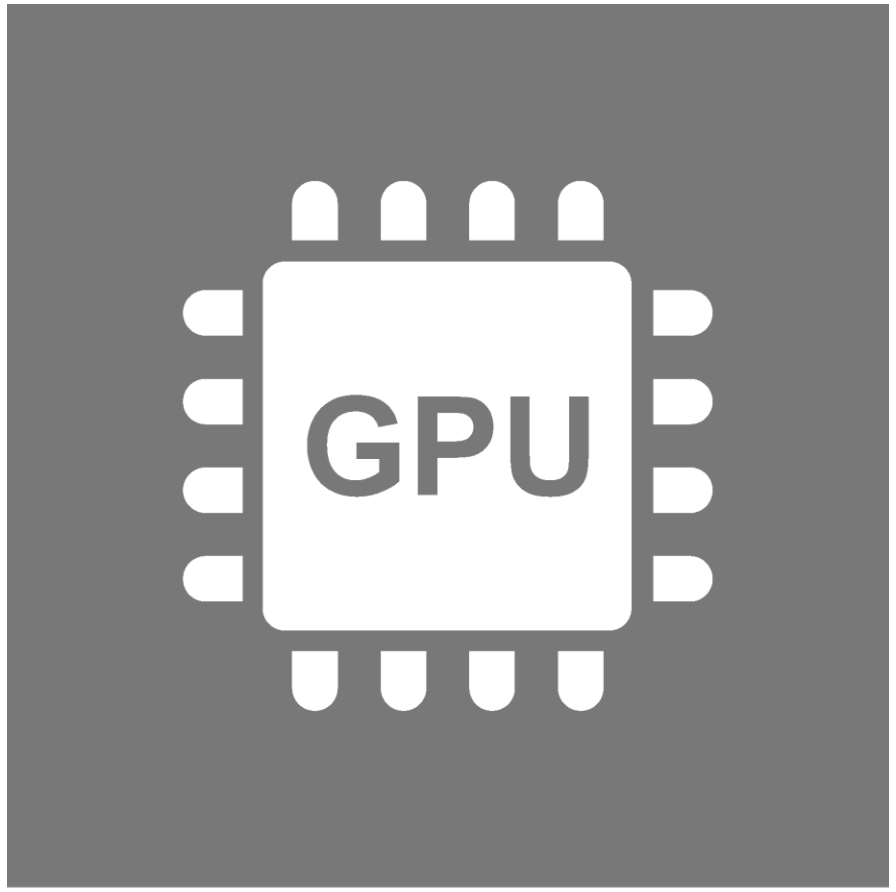
Keras provides some Deep Learning models with pre-trained weights, and one can use these to make predictions or extract features.



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.



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

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

Step 3

Compile the loss function and evaluates a set of weights

Step 4

Fit the model through backpropagation and optimization of weights 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

Predicts with the model prepared

Step 1: Import the libraries

Import statements are used to import specific classes and 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 provided 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'])
```

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

Step 4: Fit the Model

The 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

The 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 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
```

How Are Loss Functions Implemented in TensorFlow?

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

Syntax:

```
 model = keras.Sequential([
Dense(20, activation=tf.nn.relu,
input_shape=[len(train_features[0])]),
Dense(1)
])
```

Its 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.

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')
```

Discussion: Keras

Duration: 10 minutes

- What is Keras?

  Answer: Keras is an open-source deep learning library written in Python. It offers a user-friendly interface for building, training, and deploying deep learning models.

- What are the features of Keras?

  Answer: The features of Keras include a user-friendly API, modularity, support for multiple backends, built-in neural network layers, high-level abstractions, and seamless integration with other deep learning tools and libraries.

- Why is Keras used?

  Answer: Keras is used for its simplicity, user-friendly API, and seamless backend integration, making it ideal for rapid deep learning model prototyping.

Knowledge Check

Key Takeaways

- TensorFlow is a flexible, open-source library for machine learning and artificial intelligence.
- TensorFlow offers pipelining, allowing multiple neural networks to be trained simultaneously and run efficiently on large-scale models.
- TensorBoard, a suite of visualization tools, makes it easier to visualize the computational graph.
- Keras is a high-level neural network library that makes the creation of neural network models easier, particularly for deep learning and natural language processing.

Knowledge Check

1

What is 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

1

What is 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

2

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

2

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

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!