Deep Learning with Keras and TensorFlow

Deep Neural Network and Tools

Learning Objectives

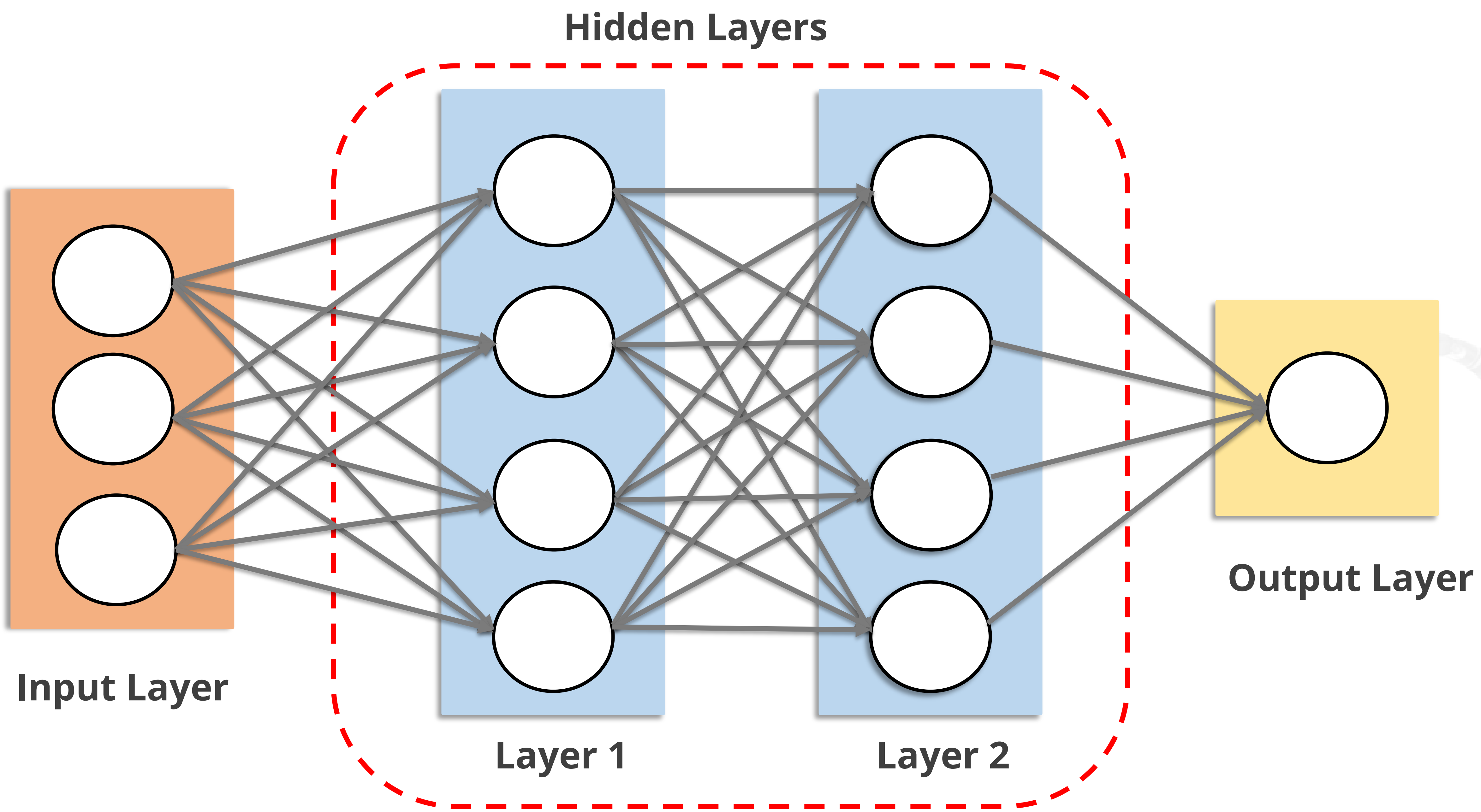
By the end of this lesson, you will be able to:

- Explain a deep neural network
- Design a deep neural network step by step
- Choose a loss function for a deep neural network
- Describe and work with deep learning tools

Deep Neural Network

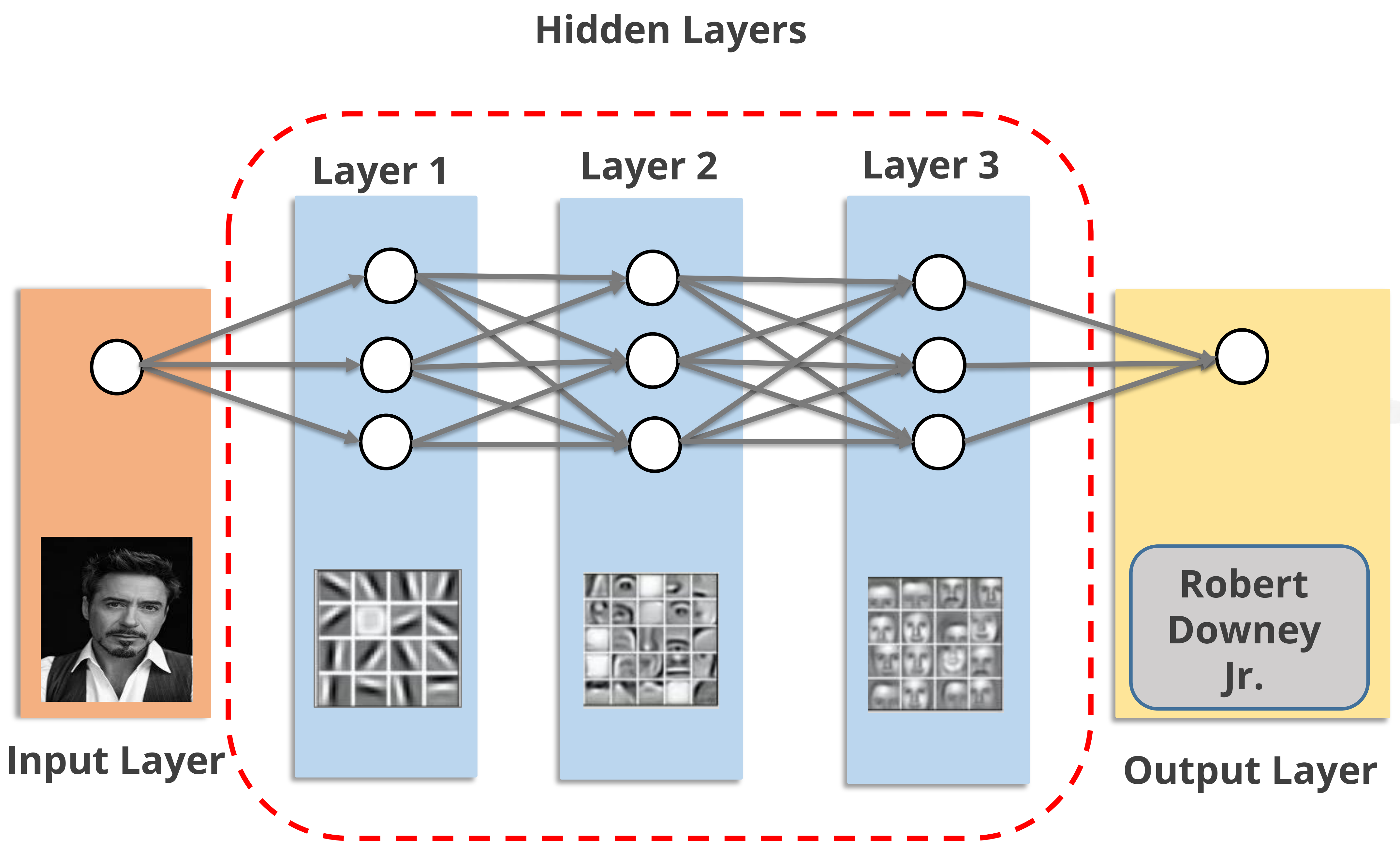
Deep Neural Network

When a neural network contains more than one hidden layer it becomes a Deep Neural Network.

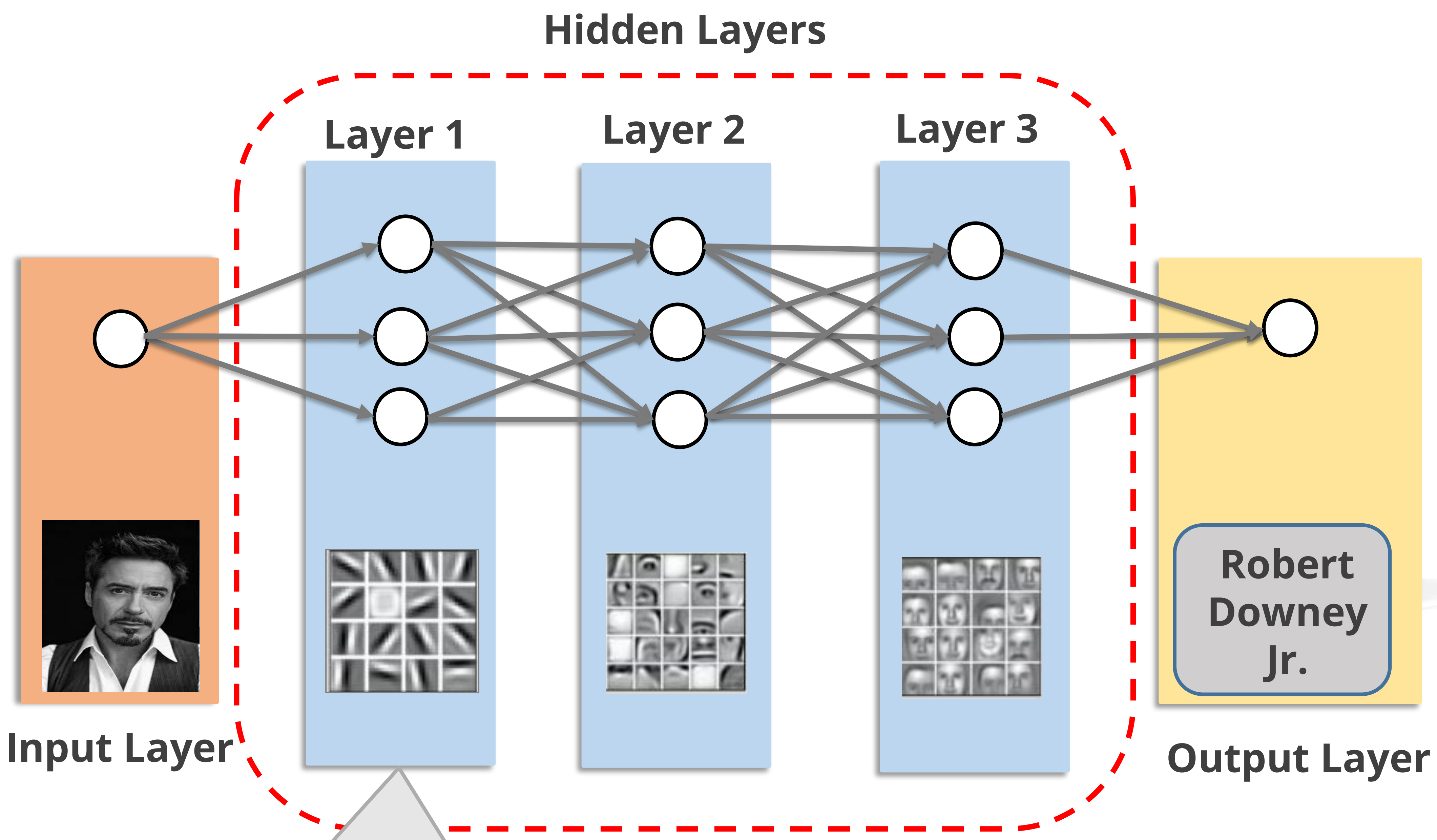


Deep Learning: Example

In deep neural network, each layer recognizes a certain set of features based on the previous layer's output.

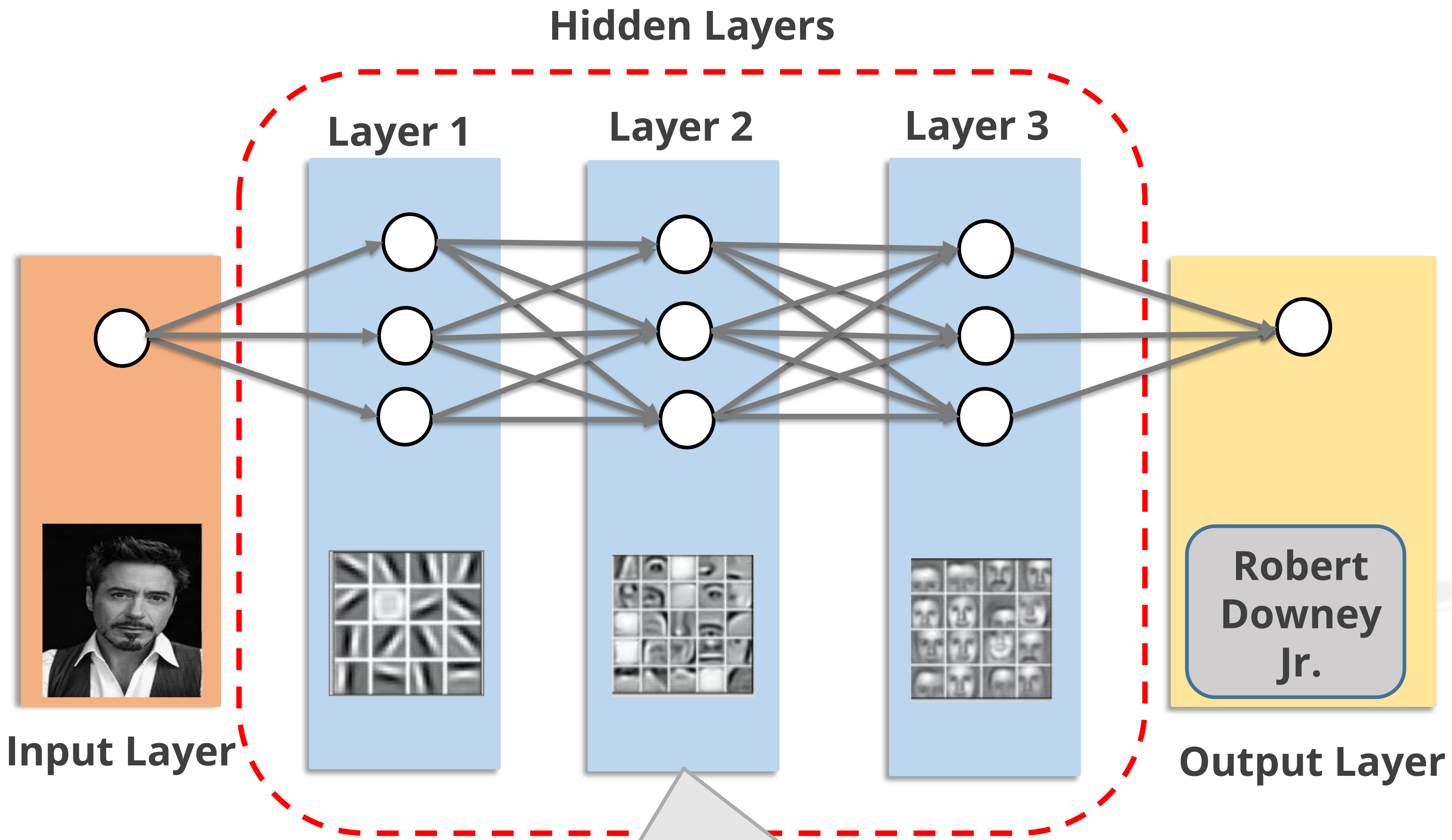


Deep Learning: Example



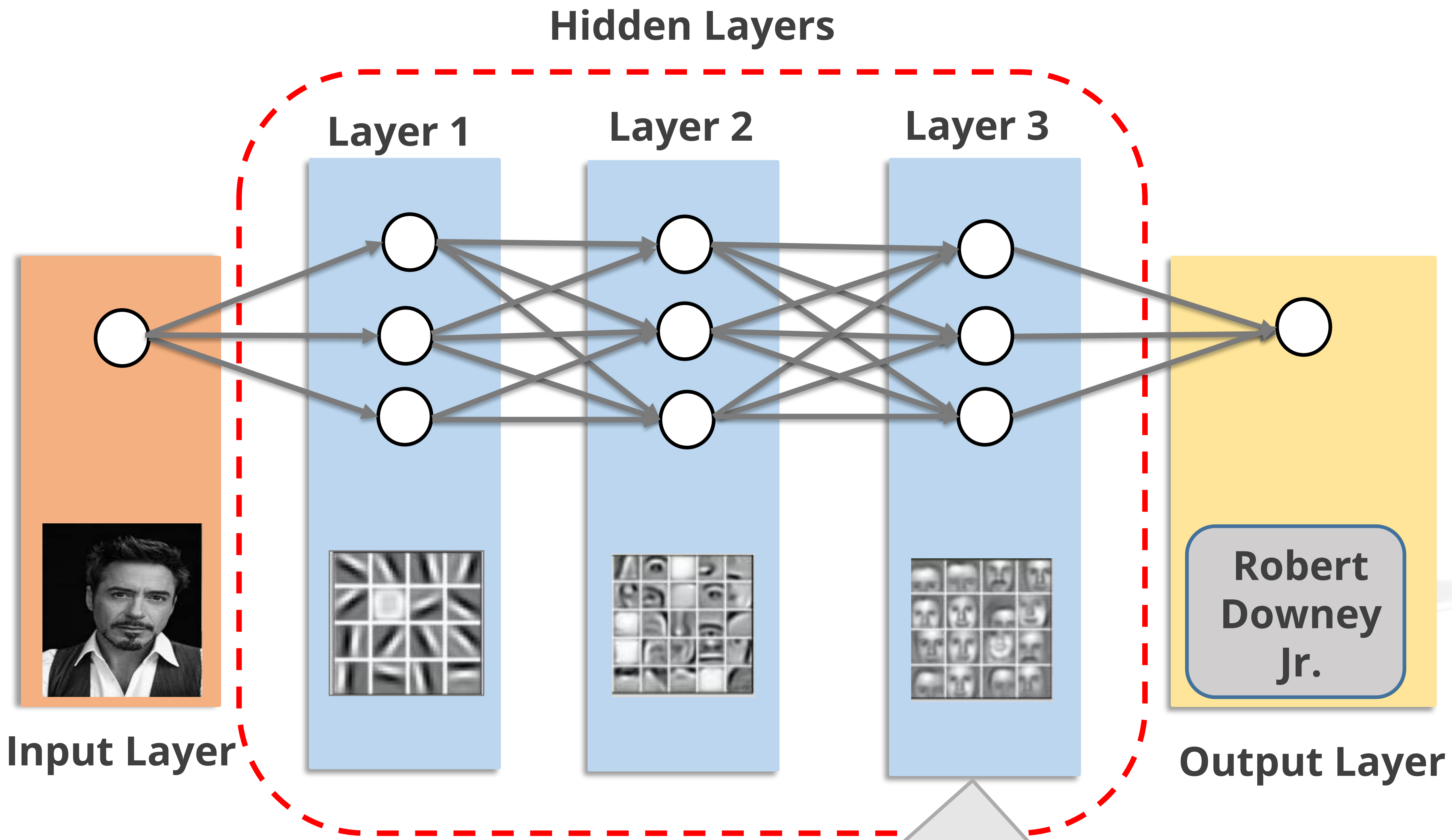
The first hidden layer trains on the input and identifies the edges.

Deep Learning: Example



The second hidden layer gets the identified edges as input and gives a combination of edges as an output.

Deep Learning: Example

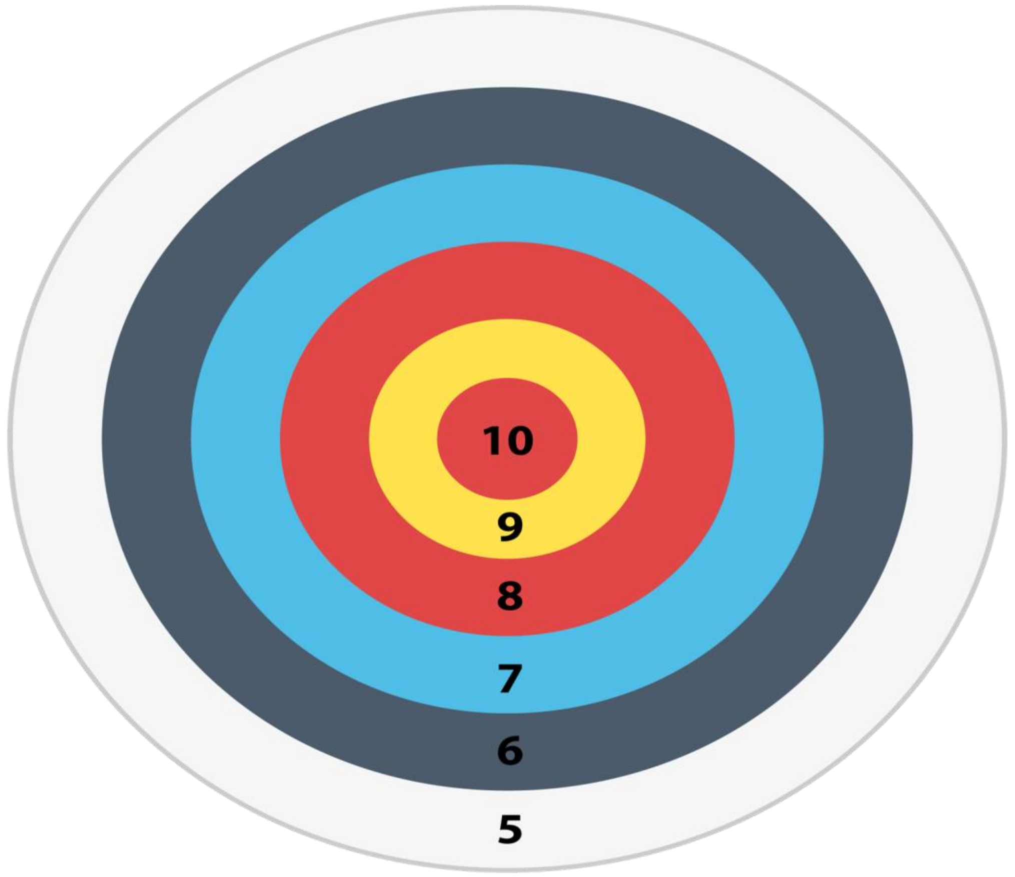


The third layer distinguishes different facial features that leads to the image recognition of the input.

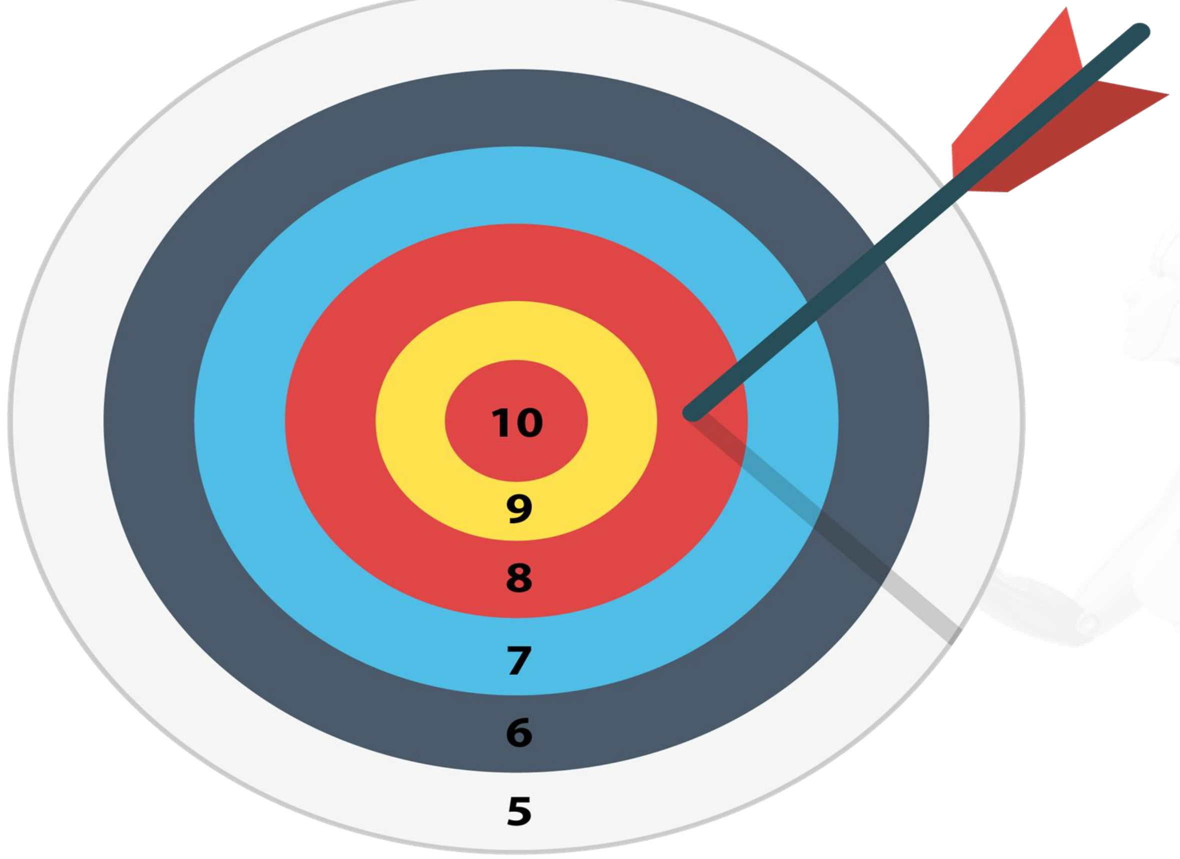
Loss Function

What Is Loss?

In a deep learning model, while predicting, the output deviates from the actual value, the quantitative measure of this difference is called loss. For example;



Here the actual value is 10

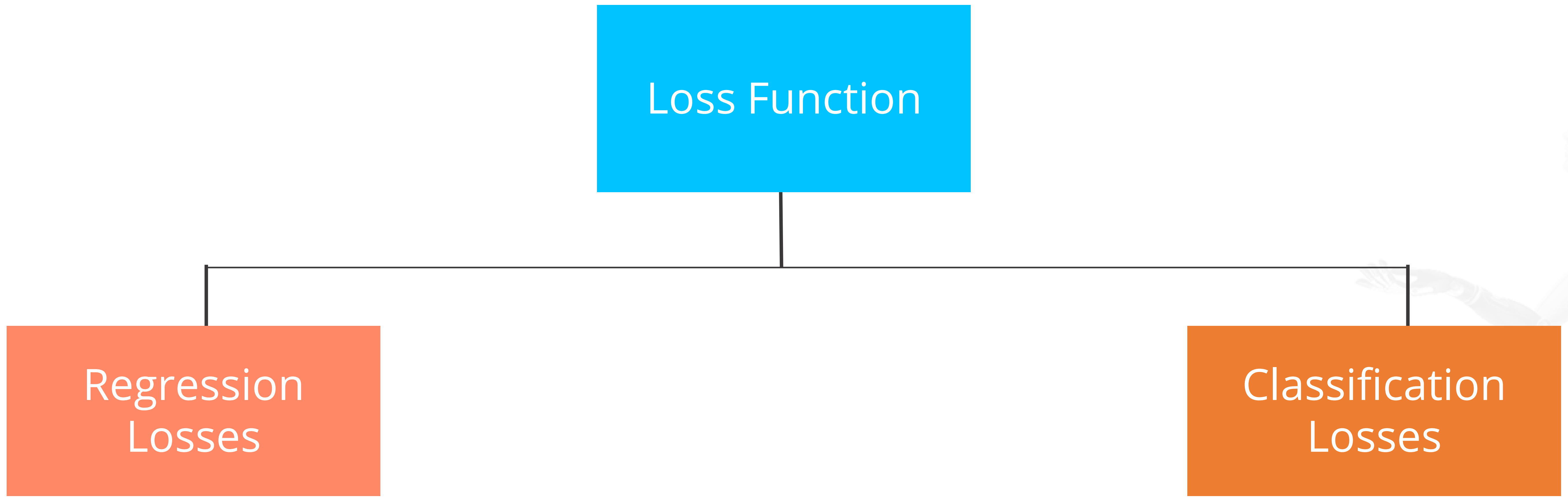


Arrow hit the circle with point 8

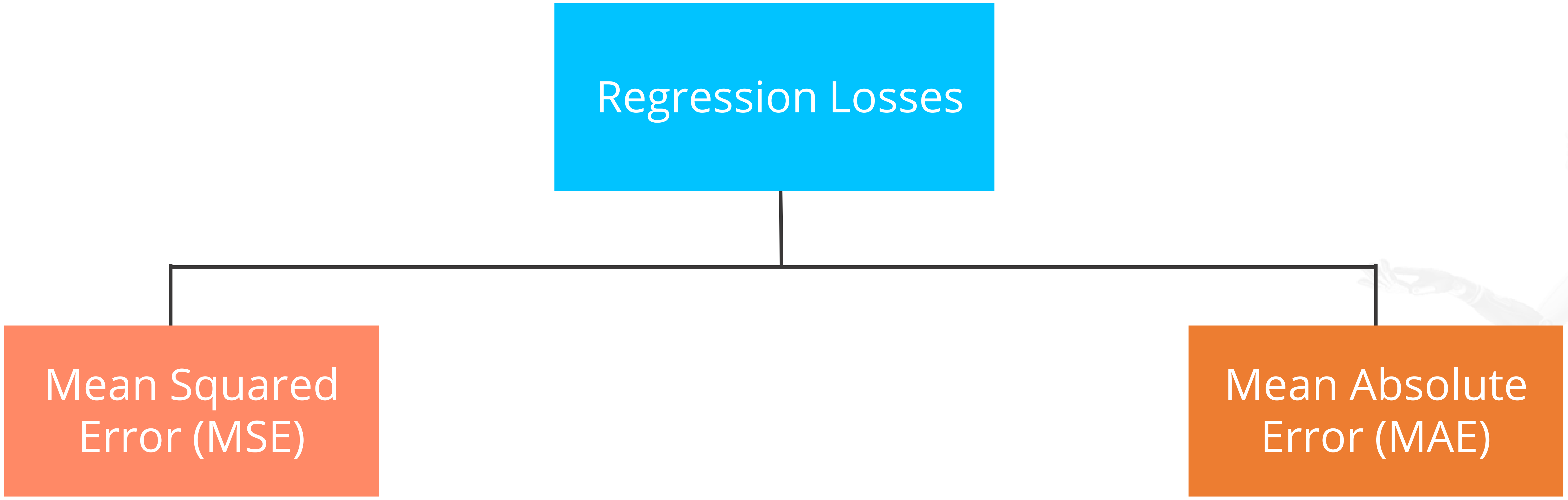
Here our loss will be, Actual value – Predicted Value, i.e., 10 – 8 = 2.

Loss Function and its Major Categories

The losses of deep learning models can be evaluated very easily by using Loss Function.



Types of Regression Losses



Mean Squared Error (MSE)

MSE is the average squared difference between actual and predicted value for N number of training data.

MSE = Sum of Squared Errors/N

Y (Actual Value)	Y′ (Predicted Value)	$(Y - Y')^2$
10.2	9.4	0.64
7.1	6.9	0.04
17.2	18.4	1.44
9.5	11.3	3.24
11.5	11.1	0.16
Sum		5.52
MSE		5.52/5 = 1.104

Mean Squared Error (MSE)

MAE is the absolute difference between actual and predicted value for N number of training data.

MSE = Sum of Mean Errors/N

Y (Actual Value)	Y’ (Predicted Value)	IY - Y’I
10.2	9.4	0.64
7.1	6.9	0.04
17.2	18.4	1.44
9.5	11.3	3.24
11.5	11.1	0.16
Sum		5.52
MSE		5.52/5 = 1.104

Mean Squared Error (MSE)

In MSE, since each error is squared, it penalizes even small differences in prediction when compared to MAE.

MSE = Sum of Mean Errors/N

Y (Actual Value)	Y’ (Predicted Value)	IY - Y’I
10.2	9.4	0.64
7.1	6.9	0.04
17.2	18.4	1.44
9.5	11.3	3.24
11.5	11.1	0.16
Sum		5.52
MSE		5.52/5 = 1.104

MSE or MAE?

In MSE, since each error is squared, it penalizes even small differences in prediction when compared to MAE.

Y (Actual Value)	Y' (Predicted Value)	$(Y - Y')^2$	$\lvert Y - Y' \rvert$
10.2	9.4	0.64	0.8
7.1	6.9	0.04	0.2
17.2	18.4	1.44	1.2
9.5	11.3	3.24	1.8
11.5	11.1	0.16	0.4
Sum		5.52	4.4
Loss Function		MSE = 5.52/5 = 1.104	MAE = 4.4/5 = 0.88

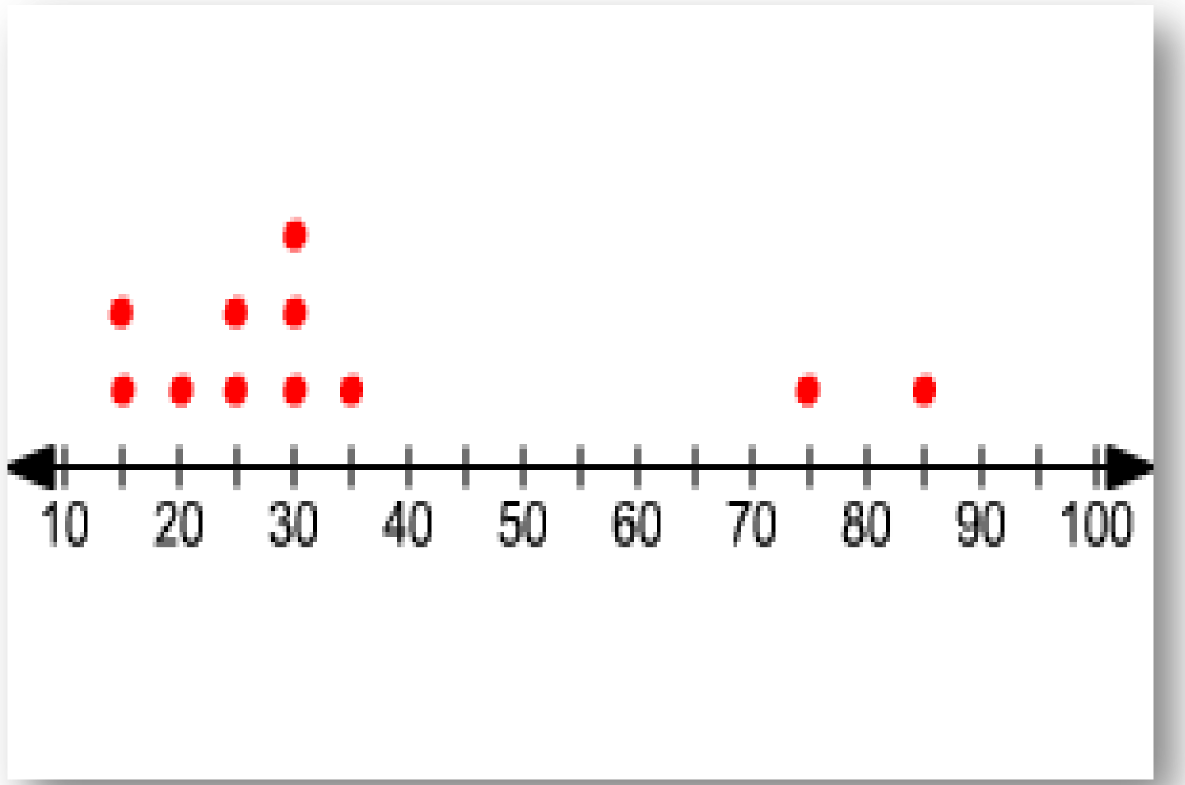
MSE or MAE?

Effect of MSE is adverse on outliers. Since each error is squared in MSE, the final MSE also increases. For example:

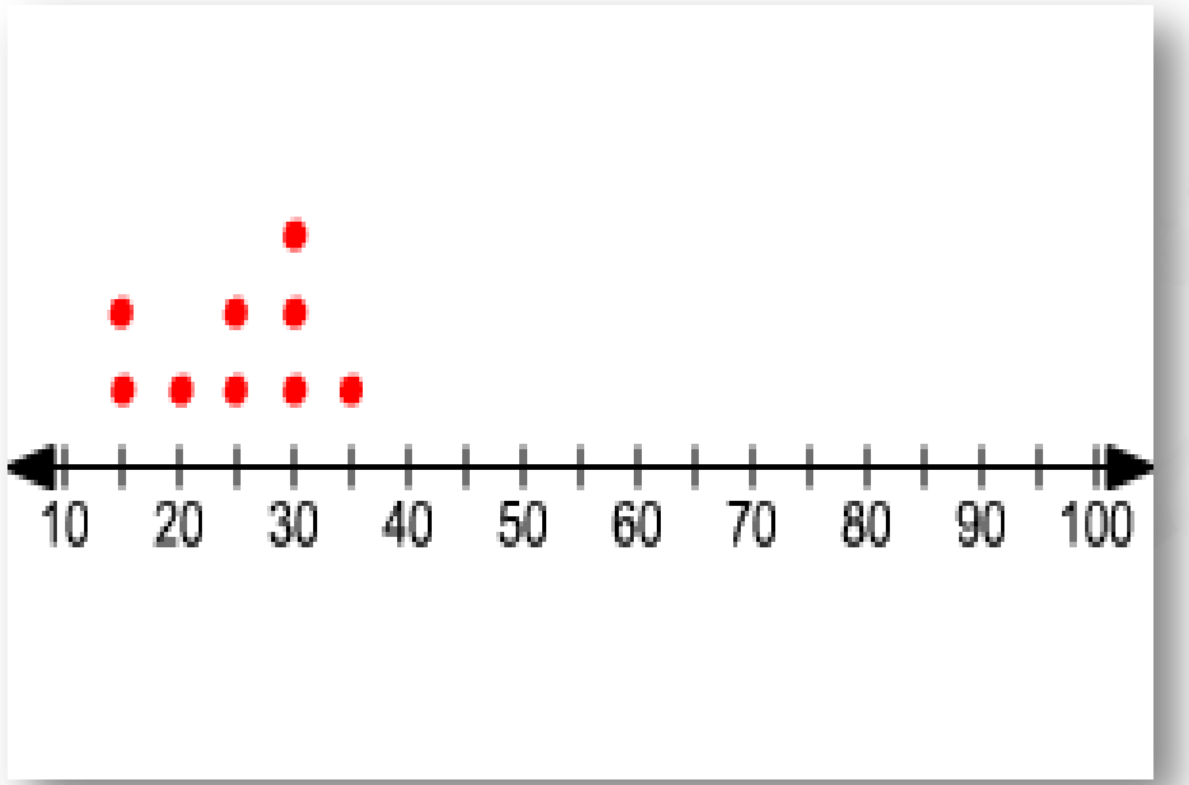
Y (Actual Value)	Y′ (Predicted Value)	$(Y - Y')^2$	IY - Y′I
10.2	9.4	0.64	0.8
7.1	6.9	0.04	0.2
17.2	18.4	1.44	1.2
31.5	11.3	408.04	20.2
11.5	11.1	0.16	0.4
Sum		5.52	4.4
Loss Function		MSE = 415.84/5 = 83.16	MAE = 27.2/5 = 5.44

MSE or MAE?

If the data has outliers, MAE will be a better option over MSE. For data without outliers MSE is preferable.

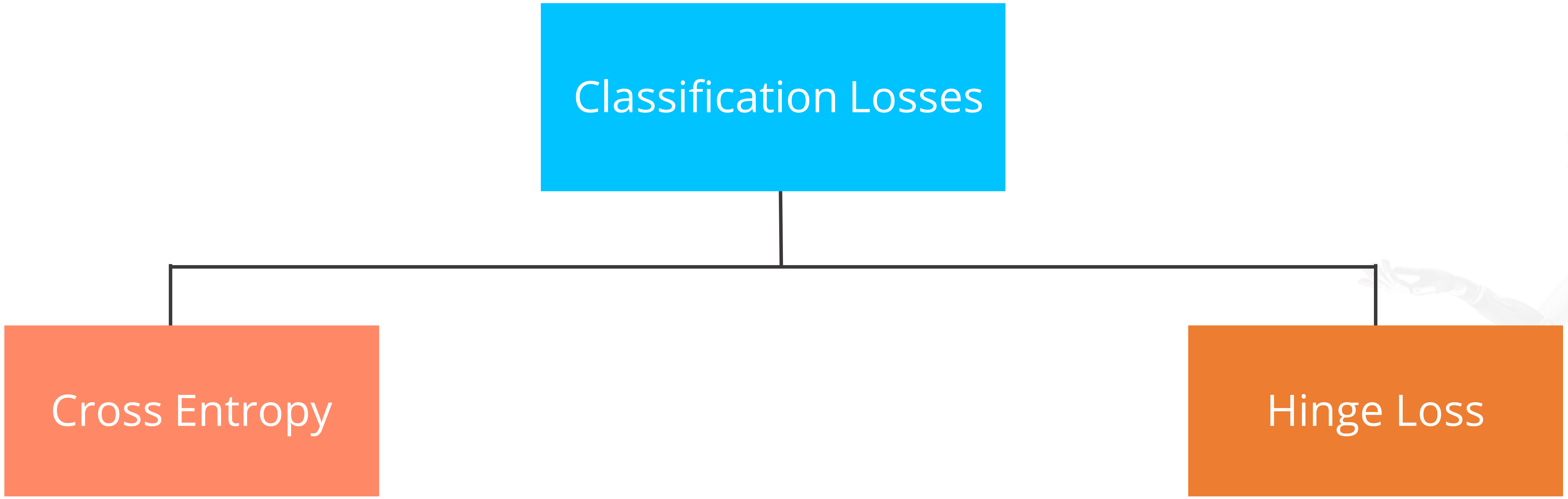


MAE as loss function



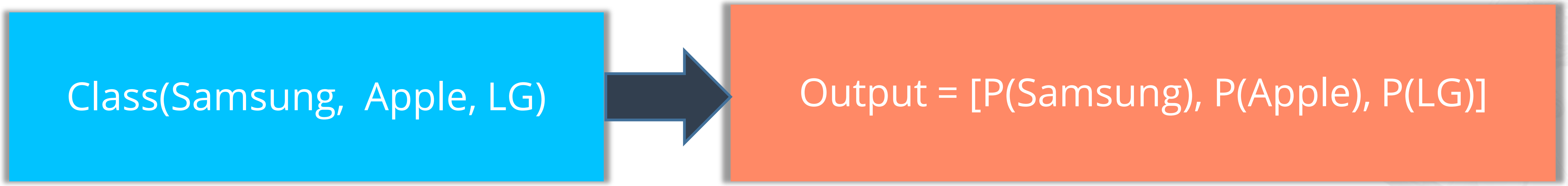
MSE as loss function

Types of Classification Losses



Cross Entropy

Cross entropy is a way to calculate distance between two probability distributions. For example, Let us consider a classification problem of 3 classes.


Class(Samsung, Apple, LG)
Output = [P(Samsung), P(Apple), P(LG)]

The class with highest probability is the winner.

Cross Entropy

If the predicted probability distribution is not close to the actual value, the model adjusts its weight.

Output = [P(Samsung), P(Apple), P(LG)]

Samsung = [1,0,0]

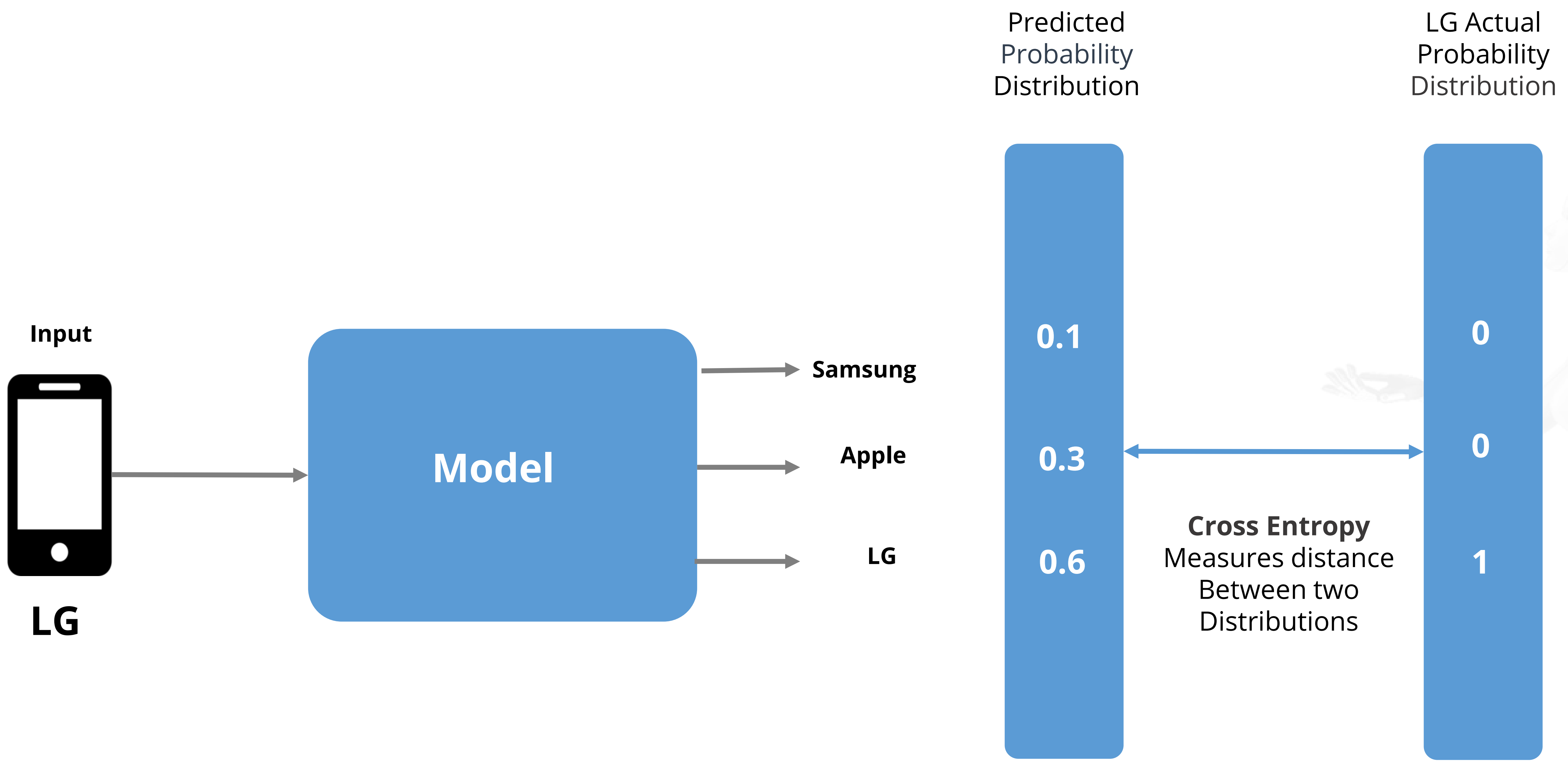
Apple = [0,1,0]

LG = [0,0,1]

The actual probability distribution for each class

Cross Entropy

In this scenario, cross entropy is used as a tool to calculate the difference predicted probability distribution from the actual one.


Input
LG
Model
Samsung
Apple
LG
Predicted Probability Distribution
0.1
0.3
0.6
LG Actual Probability Distribution
0
0
1
Cross Entropy
Measures distance Between two Distributions

Intuition behind Cross Entropy

Calculation Cross Entropy

- The model gives the probability distribution for N classes for a particular input data C.

$$P(C) = [y1' , y2' , y3' \ldots yN']$$

- The actual or target probability distribution of the data C is:

$$A(C) = [y1 , y2 , y3 \ldots yN]$$

- Cross entropy for data C is calculated as:

$$\text{CrossEntropy}(A,P) = - (y1*\log(y1') + y2*\log(y2') + y3*\log(y3') + \ldots + yN*\log(yN'))$$

Calculation Cross Entropy

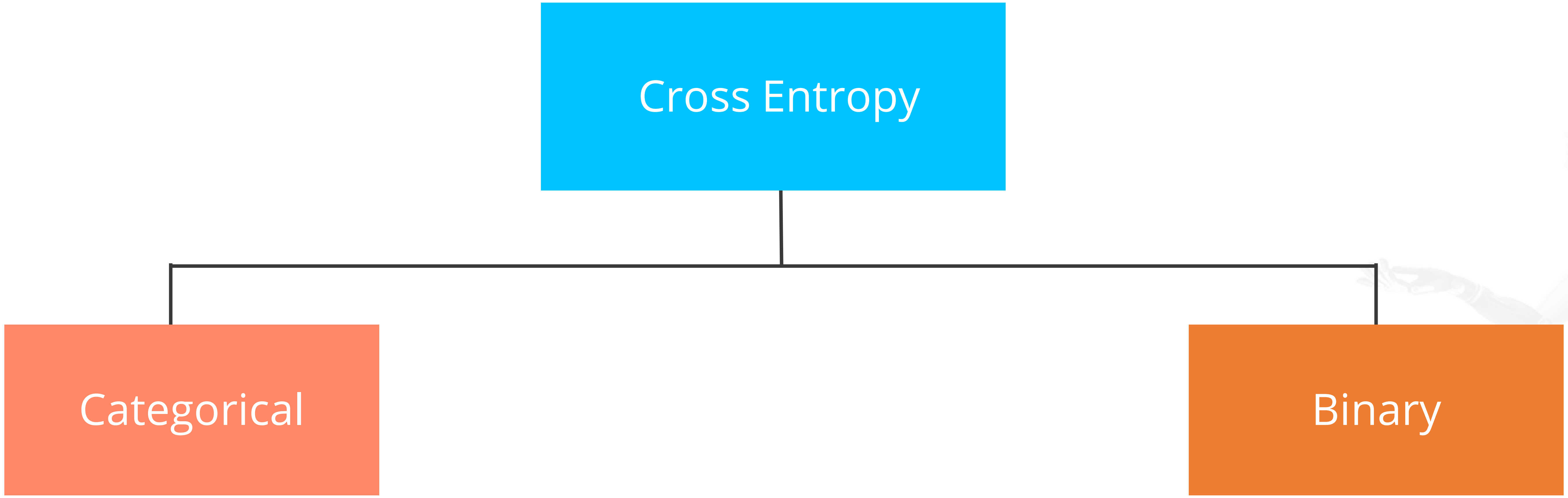
The following formula measures the cross entropy for a single observation or input data from the example:

P(LG) = [0.6, 0.3, 0.1]

A(LG) = [1, 0, 0]

CrossEntropy(A,P) = – (1*Log(0.6) + 0*Log(0.3)+0*Log(0.1)) = 0.51

Types of Cross Entropy



Categorical Cross Entropy

Categorical Cross Entropy = Sum of Cross Entropy for N data/N

Data	Actual Probability Distribution	Predicted Probability Distribution	Cross Entropy
Samsung	[1, 0, 0]	[0.6, 0.3, 0.1]	– (1*Log(0.6) + 0*Log(0.3)+0*Log(0.1)) = 0.51
Samsung	[1, 0, 0]	[0.9, 0.1, 0]	– (1*Log(0.9) + 0*Log(0.1)+0*Log(0.1)) = 0.1
Apple	[0, 1, 0]	[0.2, 0.7, 0.1]	– (0*Log(0.2) + 1*Log(0.7)+0*Log(0.1)) = 0.35
LG	[0, 0, 1]	[0.3, 0.2, 0.5]	– (0*Log(0.3) + 0*Log(0.2)+1*Log(0.5)) = 0.69
Apple	[0, 1, 0]	[0.6, 0.1, 0.3]	– (0*Log(0.6) + 1*Log(0.1)+0*Log(0.3)) = 2.3
Samsung	[1, 0, 0]	[0.5, 0.2, 0.3]	– (1*Log(0.5) + 0*Log(0.2)+0*Log(0.3)) = 0.69
LG	[0, 0, 1]	[0.1, 0.1, 0.8]	– (0*Log(0.1) + 0*Log(0.1)+1*Log(0.8)) = 0.22
Loss Function			(0.51 + 0.1 + 0.35 + 0.69 + 2.3 + 0.69 + 0.22) / 7 = 4.76

Binary Cross Entropy

- Binary cross entropy assumes a binary value of 0 or 1 to denote negative and positive class respectively, when there is only one output.

- The actual output is denoted by a single variable y, then cross entropy for a particular data C can be simplified as follows:

Cross Entropy(C) = – y*log(y') when y = 1

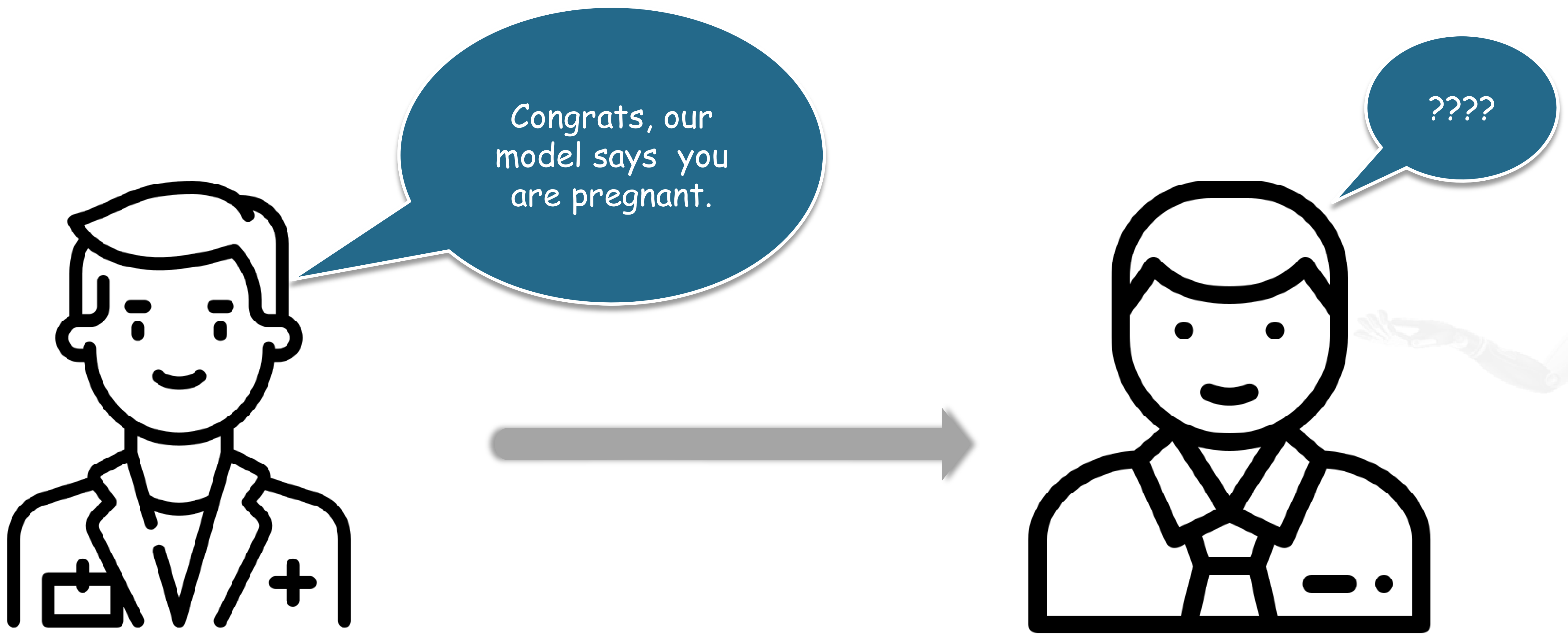
Cross Entropy(C) = – (1-y)*log(1-y') when y = 0

- The error in binary classification for complete model is given by binary cross entropy which is nothing but the mean of cross entropy for N data.

Binary Cross Entropy = Sum of Cross Entropy for N data/N

Cross Entropy over MSE/MAE

Overconfident wrong prediction occurs when MSE/MAE is used in classification, especially during the training phase.



Cross Entropy over MSE/MAE

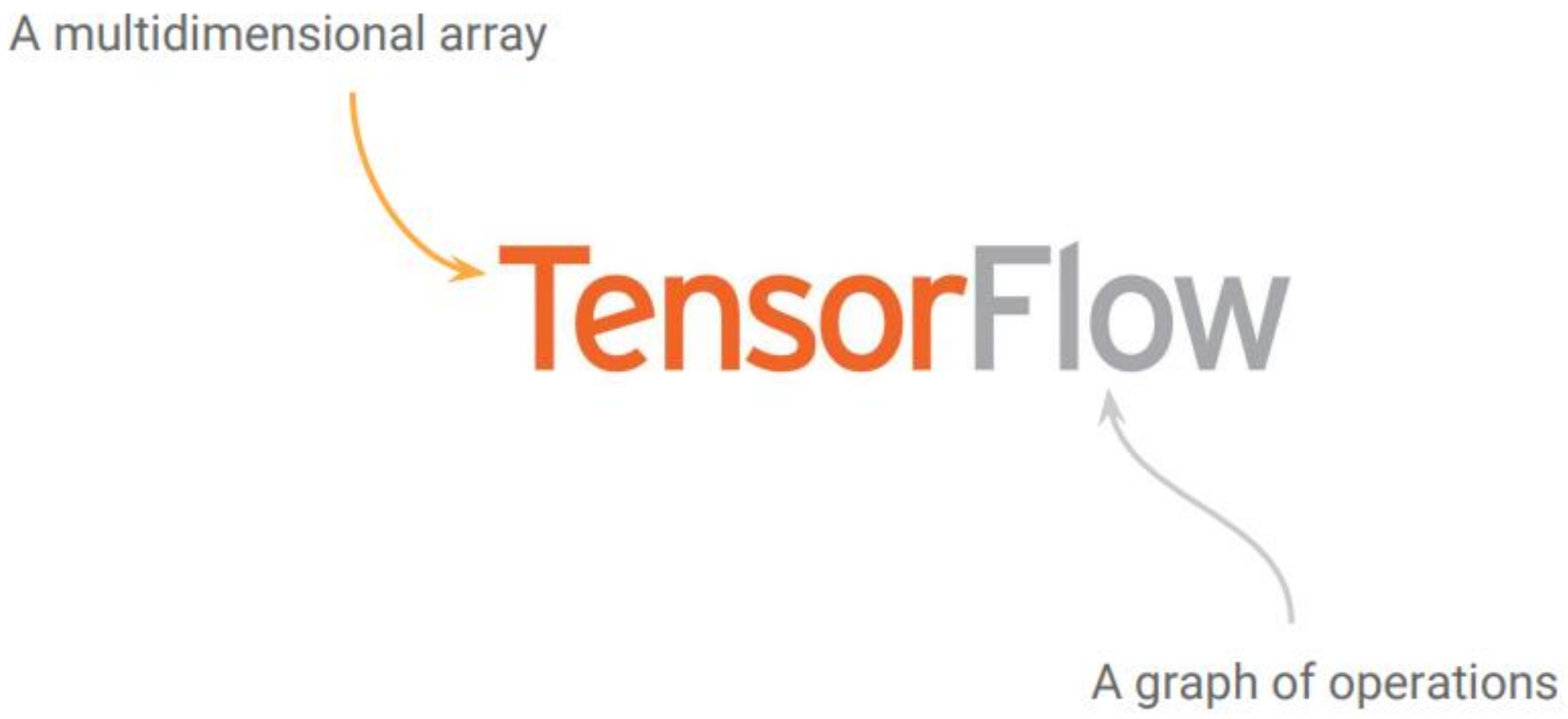
- Let us see how binary cross entropy, MAE and MSE penalizes in such situation.
- In the example below, the two scenarios of y = 1, y′ = 0.2 and y = 0, y′ = 0.8 are examples of wrong classification.

Scenario	Actual y’	Predicted y	MAE	MSE	Binary Cross Entropy
Prediction is confidently closer to actual class 1	1	0.9	I1 - 0.9I = 0.1		– 1*Log(0.9) = 0.1
Prediction is confidently closer to actual class 0	1	0.2	I1 - 0.2I = 0.8		– 1*Log(0.2) = 1.64
Prediction is confidently closer to actual class 0	0	0.1	I0 - 0.1I = 0.1		– 1*Log(1 - 0.1) = 0.1
Prediction is confidently closer to actual class 1	0	0.8	I0 - 0.8I = 0.8		– 1*Log(1 – 0.8) = 1.64

Binary Cross Entropy penalizes more severely than MAE or MSE.

TensorFlow

What Is TensorFlow?



What Is TensorFlow?

A popular open source library for deep learning and machine learning

Developed by Google Brain Team and released in 2015

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

What Is TensorFlow?

TensorFlow uses a dataflow graph to represent your computation.

Dataflow is a common programming model for parallel computing.

Benefits of Using Graph

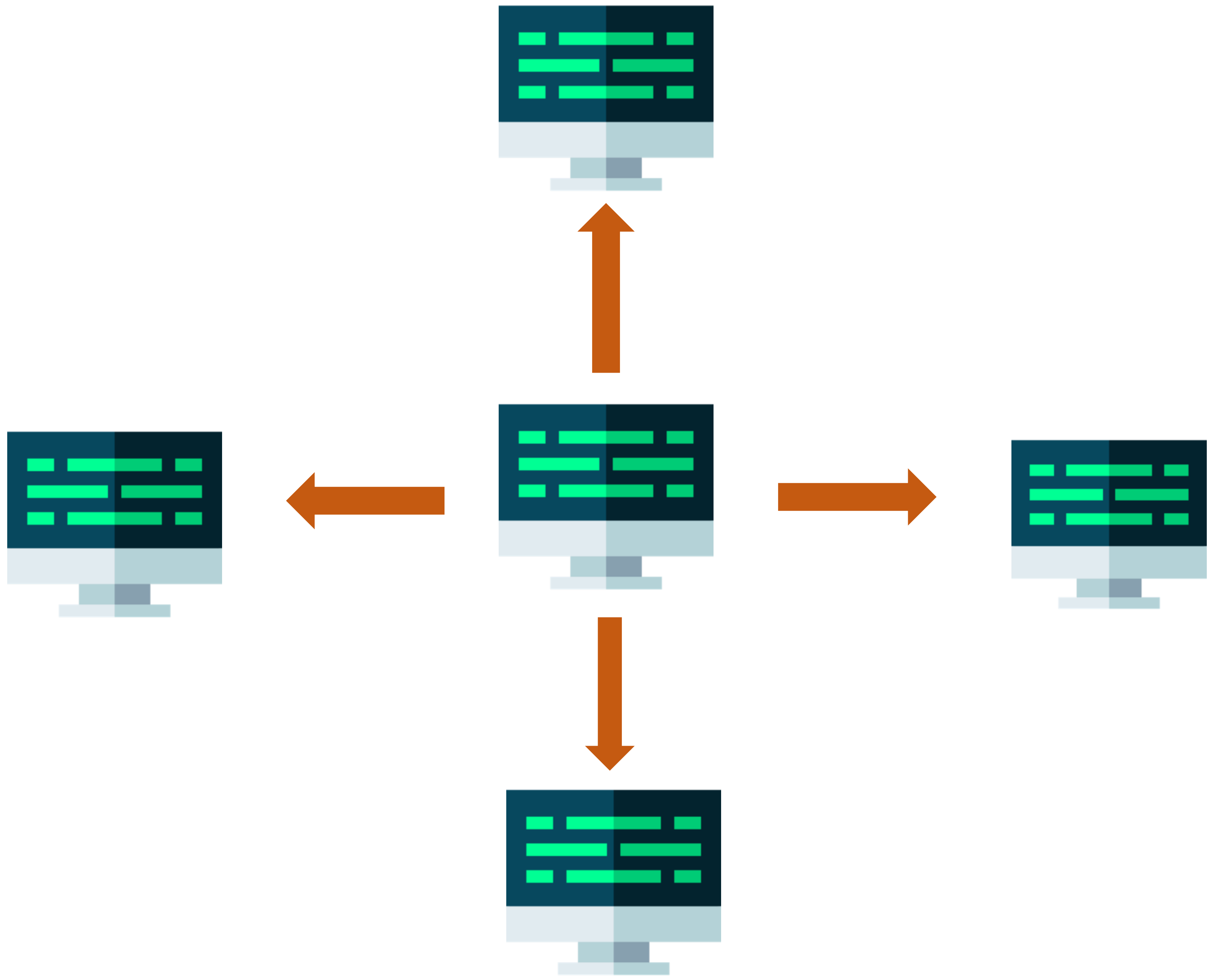
Parallelism	It is easy for the system to identify operations that can be executed parallelly.
Distributed Execution	It is possible for TensorFlow to partition your program across multiple devices CPUs, GPUs, and TPUs.
Compilation	It helps to generate faster code.
Portability	You can build a dataflow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

Why TensorFlow?

- Flexibility
- Parallel Computation
- Multiple Environment Friendly
- Large Community

TensorFlow: Parallel Computation

TensorFlow supports distributed computing.



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: 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

Installation of TensorFlow

TensorFlow 2 packages require a pip version >19.0.

- pip install --upgrade pip

DATA AND
ARTIFICIAL INTELLIGENCE

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 in order to facilitate and speed up experimentations, while remaining fully transparent and compatible with it.

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 CPU or GPU

Installation of TFLearn

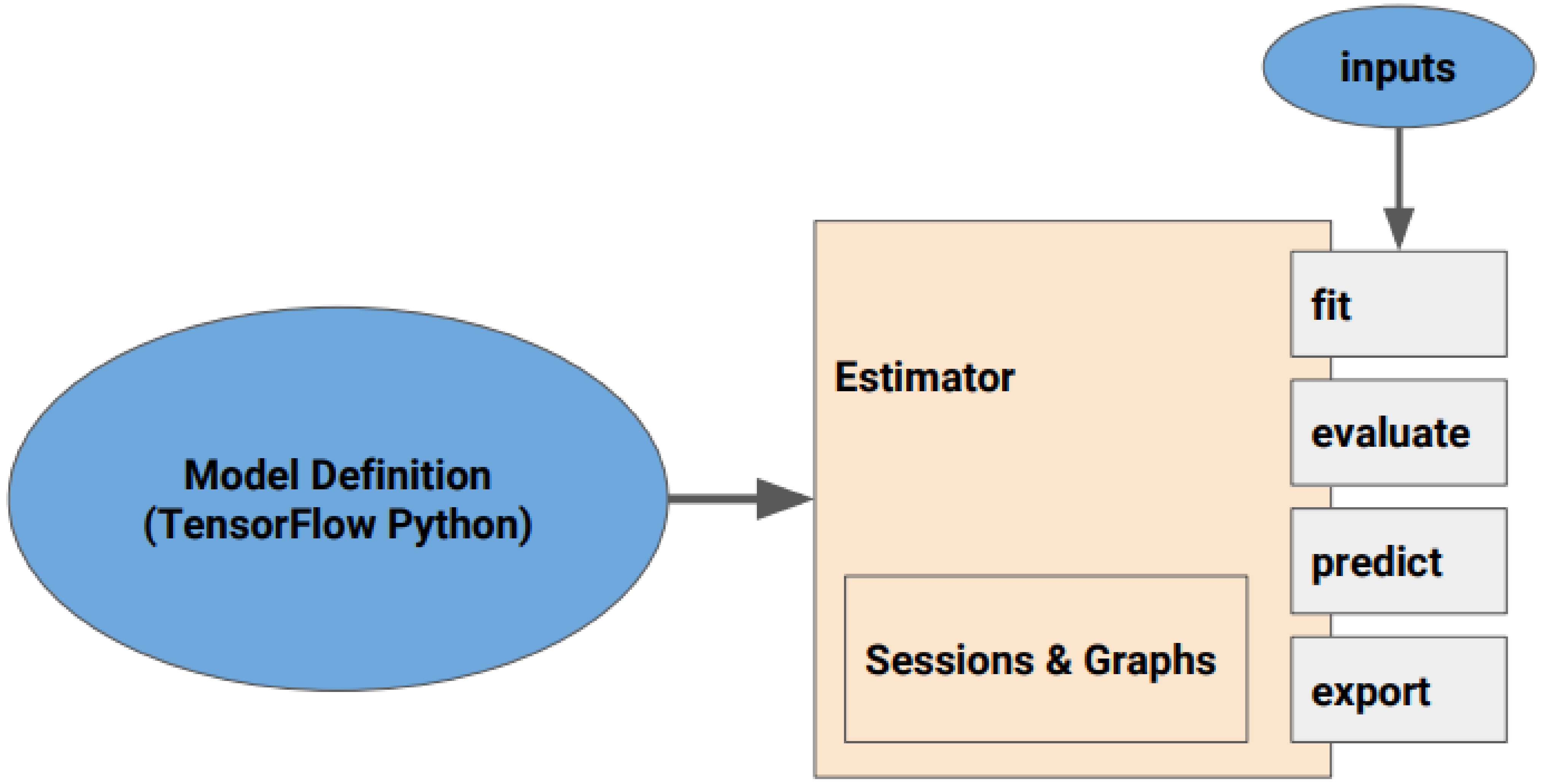
For the bleeding edge version:

- pip install git+https://github.com/tflearn/tflearn.git

For the latest stable version:

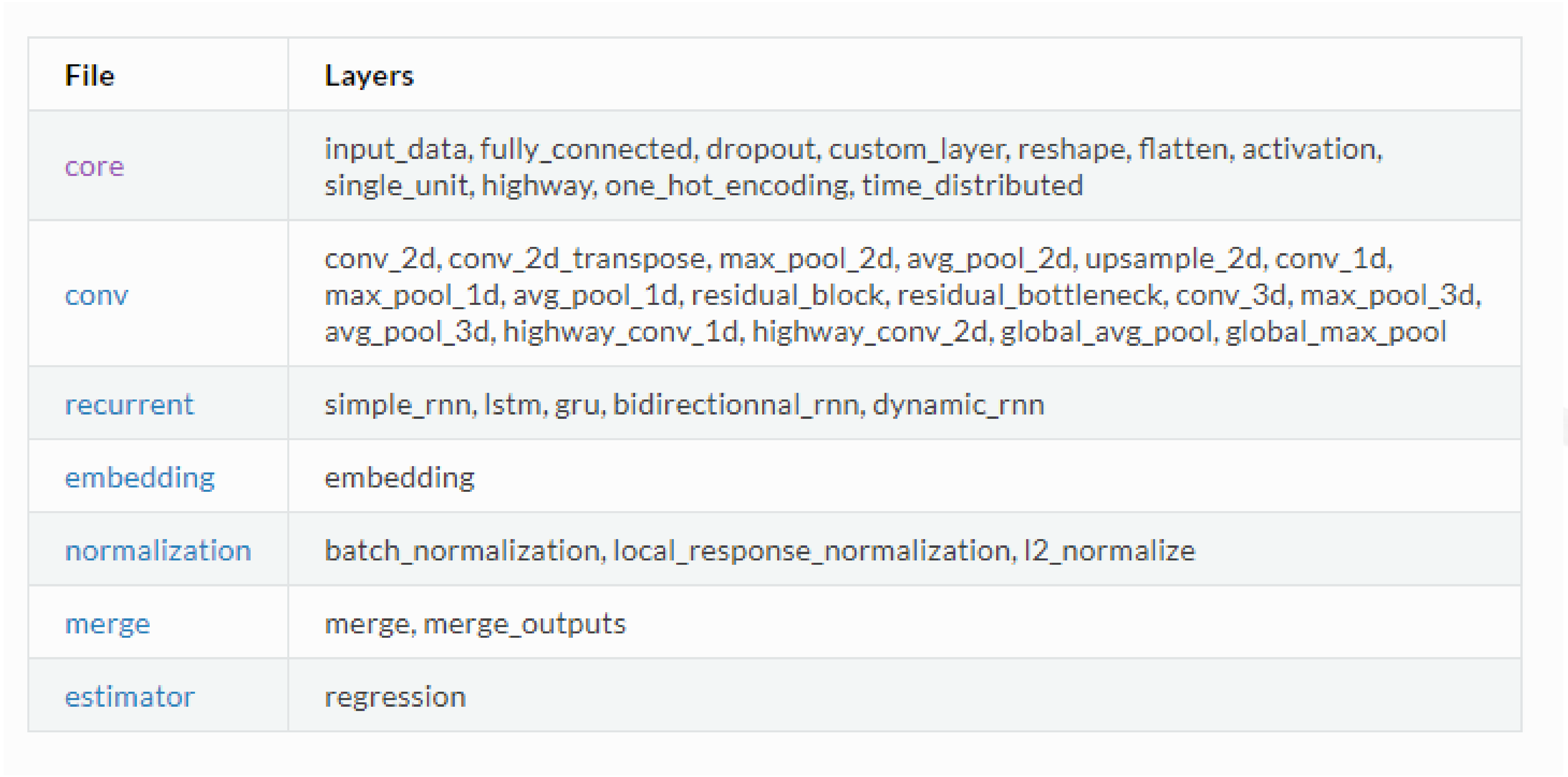
- pip install tflearn

TFLearn Model



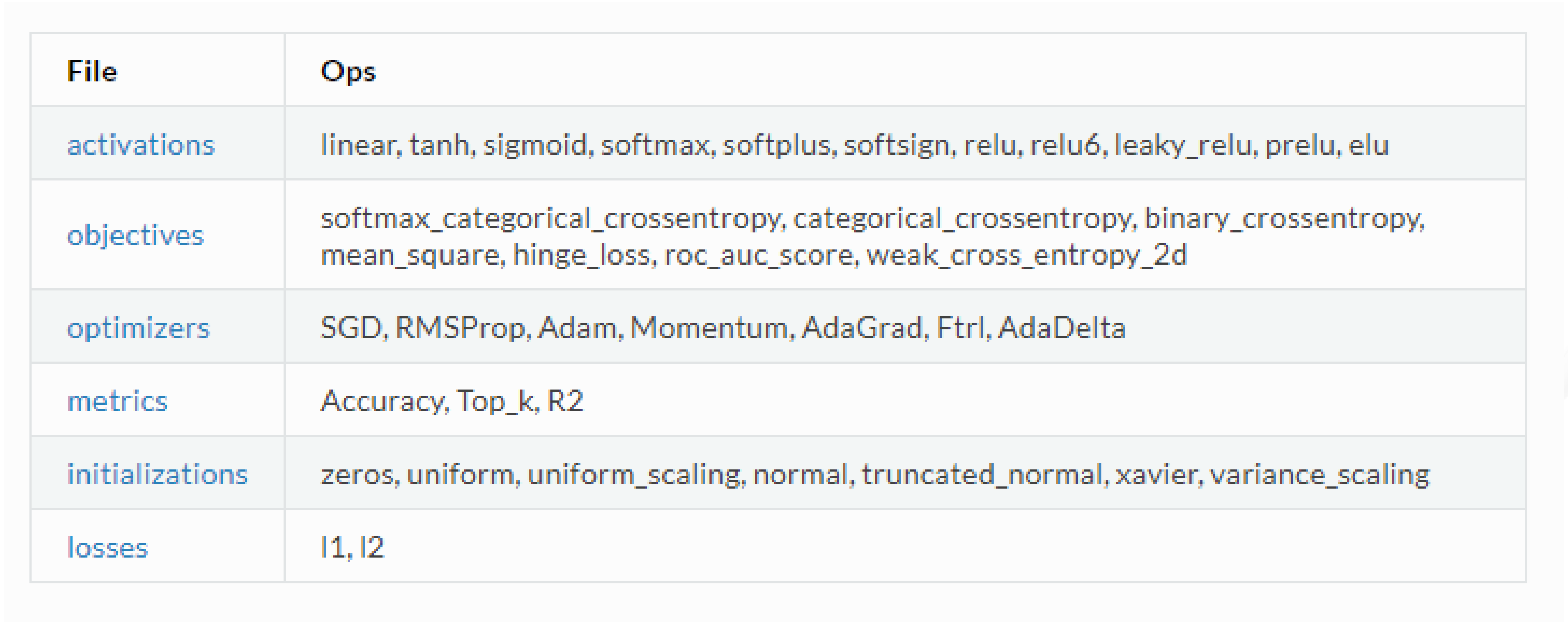
Layers of TFLearn

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, bidirectionnal_rnn, dynamic_rnn
embedding	embedding
normalization	batch_normalization, local_response_normalization, l2_normalize
merge	merge, merge_outputs
estimator	regression

Built-In Operations of TFlearn



File	Ops
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

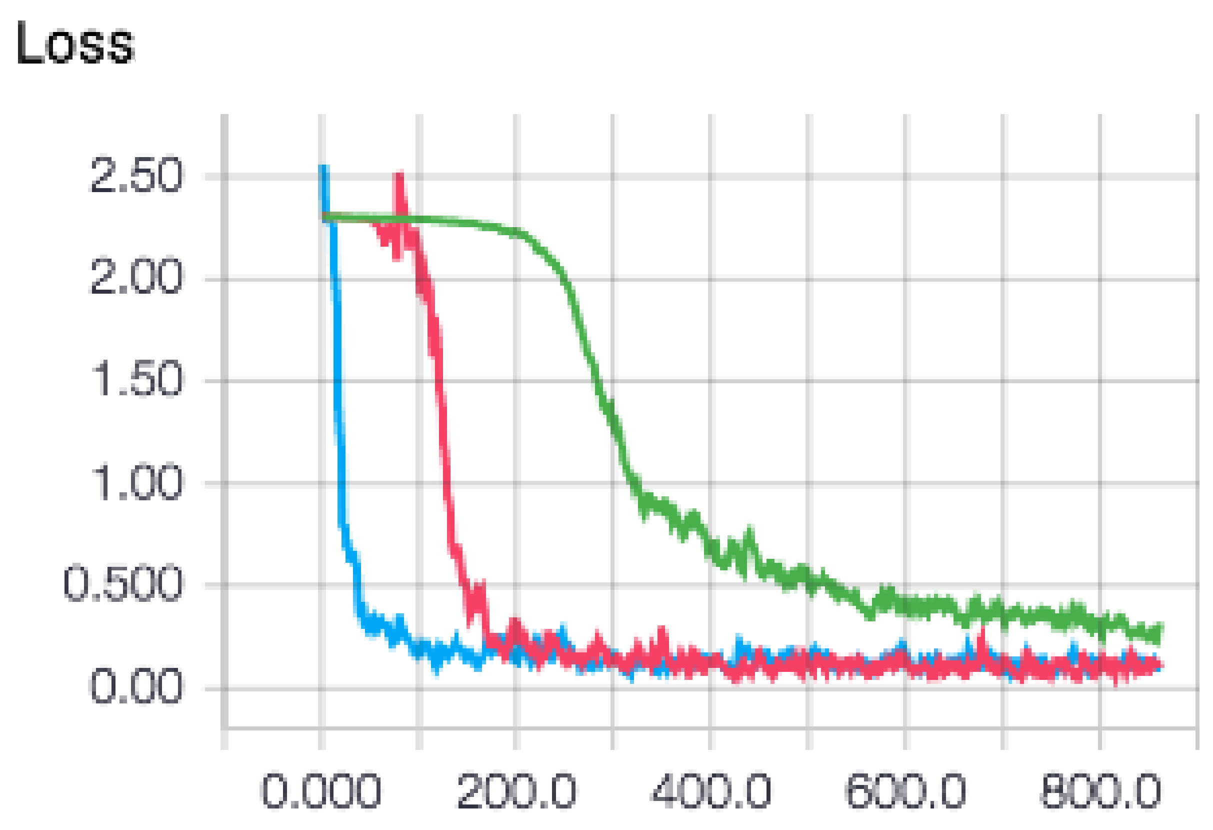
Training functions are another core feature of TFLearn. In Tensorflow, there are no prebuilt API to train a network, so TFLearn integrates a set of functions that can easily handle any neural network training, for any number of inputs, outputs, and optimizers.

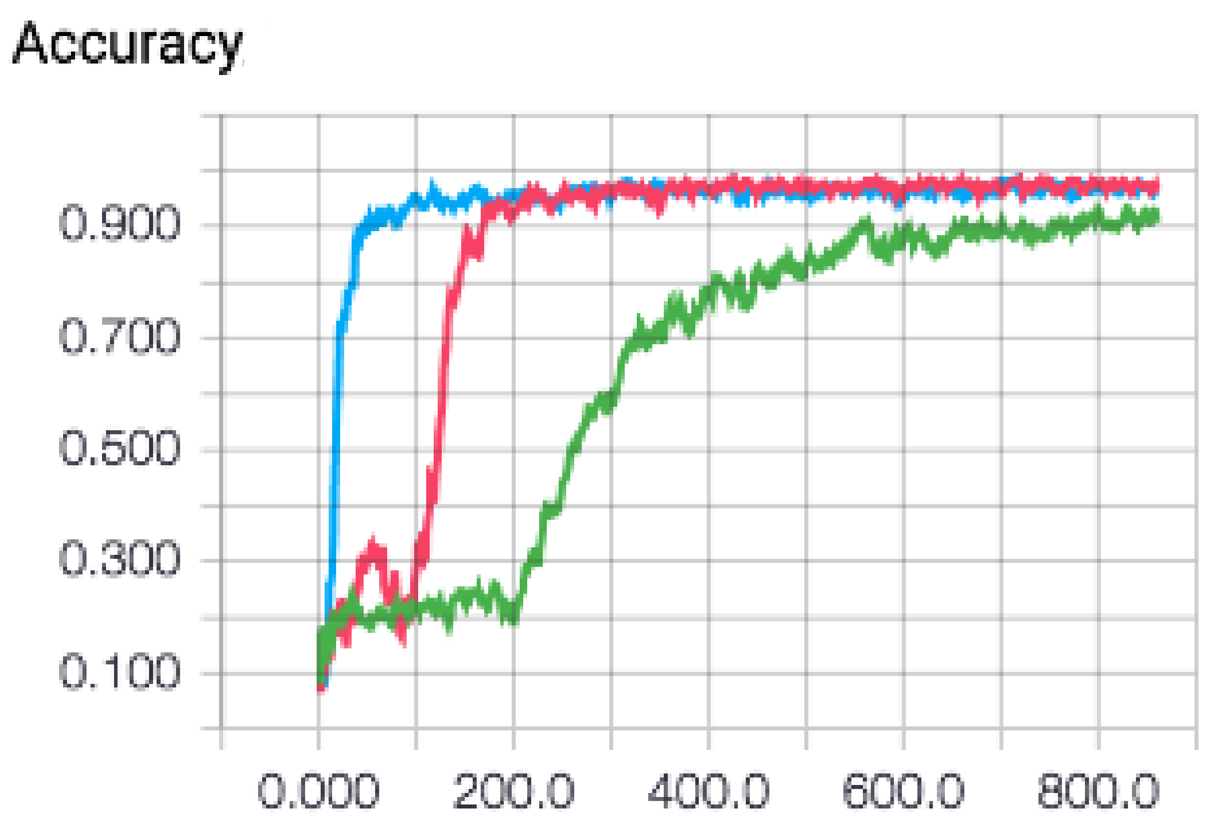
Visualization

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

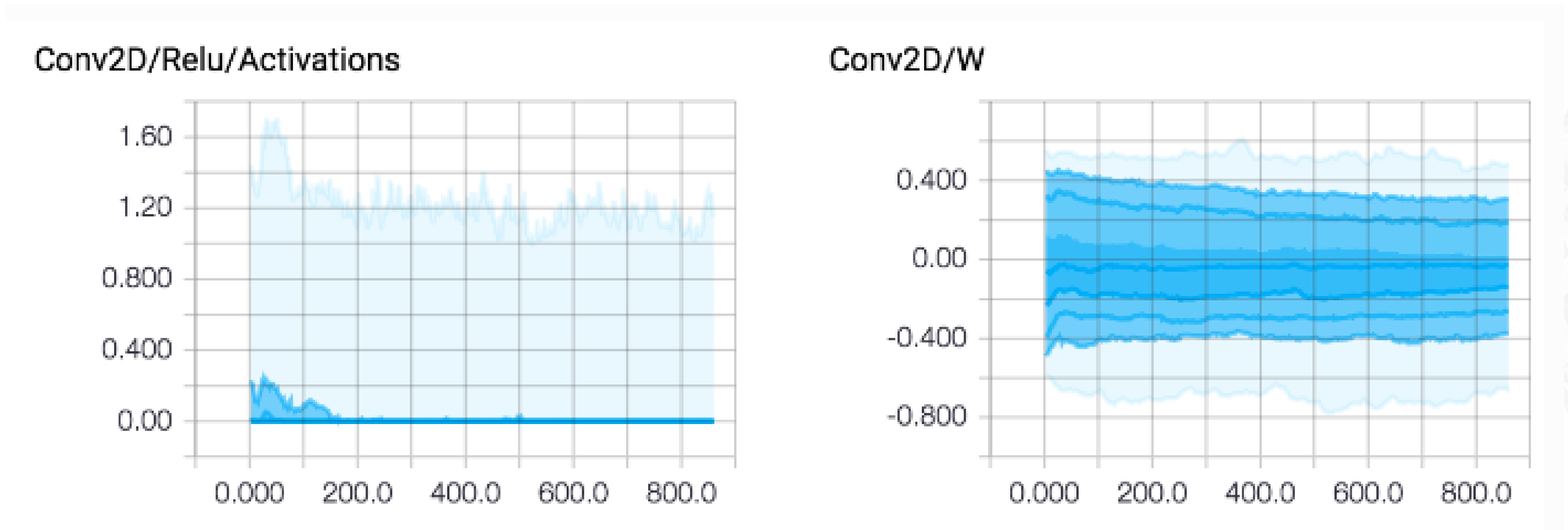
- 1: Loss and Metric (Best Speed)
- 2: Loss, Metric, and Gradients
- 3: Loss, Metric, Gradients, and Weights
- 4: Loss, Metric, Gradients, Weights, Activations, and Sparsity (Best Visualization)

Visualization: Loss and Accuracy

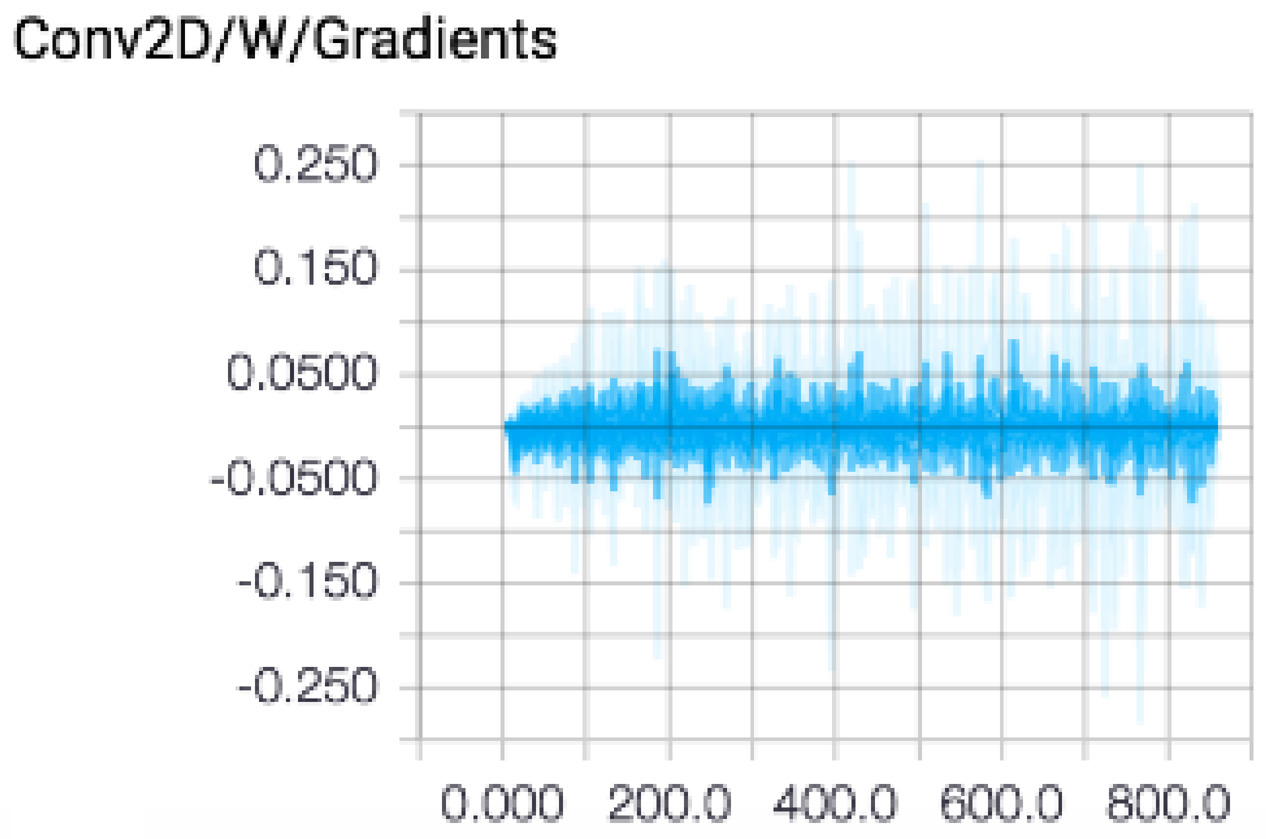


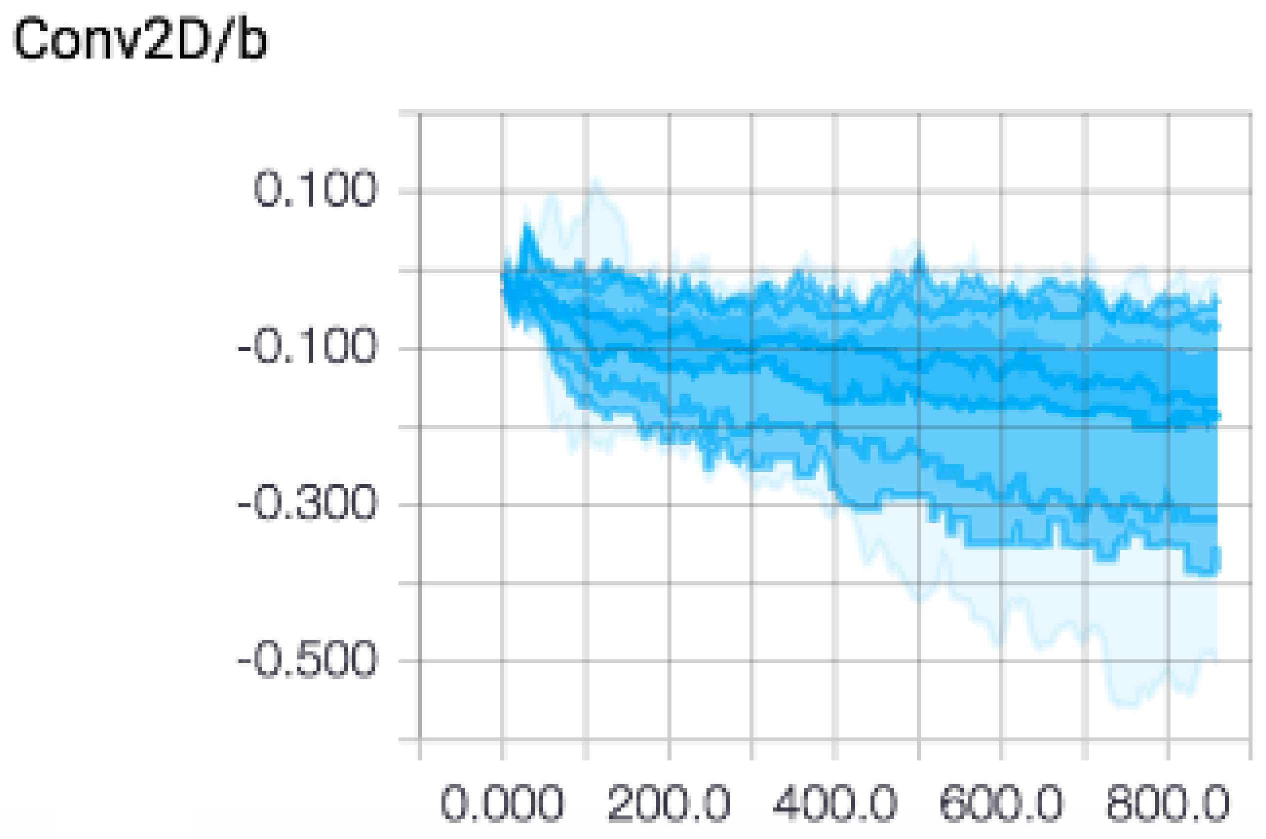


Visualization: Layers



Visualization: Layers





Weights Persistence

To save or restore a model, use 'save' or 'load' method of DNN model class.

Code

```
# Save a model
model.save('my_model.tflearn')
# Load a model
model.load('my_model.tflearn')
```

Weights Persistence

Retrieving a layer variable can either be done using the layer name, or directly by using 'W' or 'b' attributes that are supercharged to the layer's returned tensor.

Code

```
# 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]
```

Weights Persistence

To get or set the value of these variables, TFLearn model classes implement **get_weight** and **set_weights** methods:

Code

```
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))
```

simplilearn

Fine-Tuning

While defining a model in TFLearn, you can specify the layer's weights while loading the pre-trained model. This can be handled by the **restore** argument of layer functions and it is only available for layers with weights.

Code

```
# Weights will be restored by default.
fc_layer = tflearn.fully_connected(input_layer, 32)
# Weights will not be restored, if specified so.
fc_layer = tflearn.fully_connected(input_layer, 32, restore='False')
```

Data Management

TFLearn supports numpy array data. Additionally, it supports HDF5 for handling large datasets. TFLearn can directly use HDF5-formatted data:

Code

```
# Load hdf5 dataset
h5f = h5py.File('data.h5', 'r')
X, Y = h5f['MyLargeData']

... define network ...

# Use HDF5 data model to train model
model = DNN(network)
model.fit(X, Y)
```

Data Preprocessing and Augmentation

TFLearn provides wrappers to easily handle data preprocessing and data augmentation. TFLearn data stream is designed with computing pipelines in order to speedup training by pre-processing data on CPU while GPU is performing model training.

Code

```
# Load hdf5 dataset
h5f = h5py.File('data.h5', 'r')
X, Y = h5f['MyLargeData']

... define network ...

# Use HDF5 data model to train model
model = DNN(network)
model.fit(X, Y)
```

Data Preprocessing and Augmentation

Code

```
# Real-time image preprocessing
img_prep = tflearn.ImagePreprocessing()
# Zero Center (With mean computed over the whole dataset)
img_prep.add_featurewise_zero_center()
# STD Normalization (With std computed over the whole dataset)
img_prep.add_featurewise_stdnorm()
```

Data Preprocessing and Augmentation

Code

```
# Real-time data augmentation
img_aug = tflearn.ImageAugmentation()
# Random flip an image
img_aug.add_random_flip_leftright()

# Add these methods into an 'input_data' layer
network = input_data(shape=[None, 32, 32, 3],
                     data_preprocessing=img_prep,
                     data_augmentation=img_aug)
```

Scopes and Weights Sharing

All layers are built over **variable_op_scope**, that makes them easy to share the variables among multiple layers and make TFLearn suitable for distributed training.

Code

```
# Define a model builder
def my_model(x):
    x = tflearn.fully_connected(x, 32, scope='fc1')
    x = tflearn.fully_connected(x, 32, scope='fc2')
    x = tflearn.fully_connected(x, 2, scope='out')

# 2 different computation graphs but sharing the same weights
with tf.device('/gpu:0'):
    # Force all Variables to reside on the CPU.
    with tf.arg_scope([tflearn.variables.variable], device='/cpu:0'):
        model1 = my_model(placeholder_X)
```

Scopes and Weights Sharing

All layers with inner variables support a **scope** argument to place variables under layers with same scope name and these layers share the same weights.

Code

```
# Reuse Variables for the next model
tf.get_variable_scope().reuse_variables()
with tf.device('/gpu:1'):
    with tf.arg_scope([tflearn.variables.variable], device='/cpu:0'):
        model2 = my_model(placeholder_X)

# Model can now be trained by multiple GPUs (see gradient averaging)
```

Graph Initialization

It is useful to limit resources, or assign more or less GPU RAM memory while training. To do so, a graph initializer can be used to configure a graph by running the following:

Code

```
tflearn.init_graph(set_seed=8888, num_cores=16, gpu_memory_fraction=0.5)
```

Extending TensorFlow

TFLearn is a very flexible library designed to let you use any of its component independently. A model can be succinctly built using any combination of Tensorflow operations and TFLearn built-in layers and operations. The following are the two basic fields where TensorFlow is extended:

- Layers
- Built-In Operations

Extending TensorFlow: Layers

Any layer can be used with any other tensor from Tensorflow, i.e. you can directly use TFLearn wrappers into your own Tensorflow graph.

Code

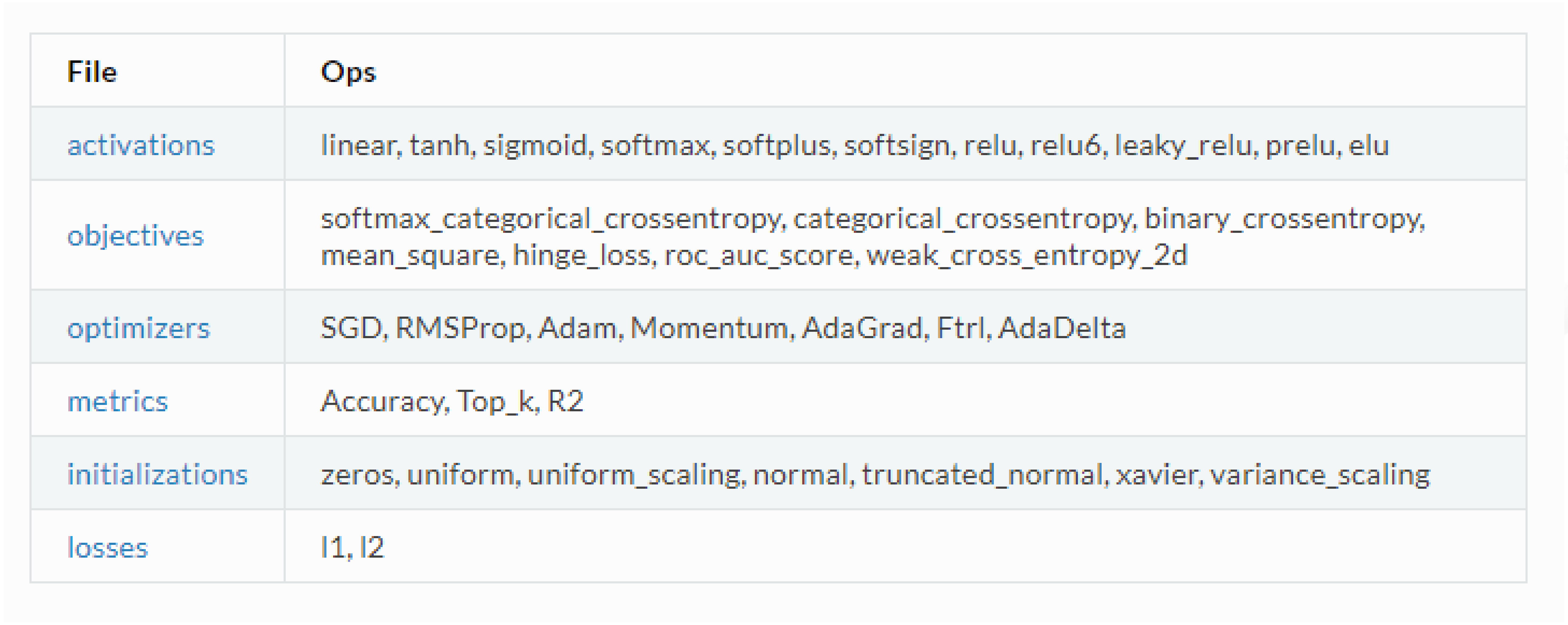
```
# Some operations using Tensorflow.
X = tf.placeholder(shape=(None, 784), dtype=tf.float32)
net = tf.reshape(X, [-1, 28, 28, 1])

# Using TFLearn convolution layer.
net = tflearn.conv_2d(net, 32, 3, activation='relu')

# Using Tensorflow's max pooling op.
net = tf.nn.max_pool(net, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME')
```

Extending TensorFlow: Built-In Operations

TFLearn built-in operations make Tensorflow graph writing faster and more readable. These are fully compatible with any TensorFlow expression. The following code examples show how to use them along with pure Tensorflow API.



File	Ops
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

Trainer, Evaluator, and Predictor

TFLearn provides some **helpers** function that can train any Tensorflow graph. It is suitable to make training more convenient, by introducing real-time monitoring, batch sampling, moving averages, tensorboard logs, data feeding, etc. It supports any number of inputs, outputs, and optimization ops.

- TFLearn implements a **TrainOp** class to represent an optimization process (i.e. backprop). It is defined as follows:

Code

```
trainop = TrainOp(net=my_network, loss=loss, metric=accuracy)
```

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.

Code

```
model = Trainer(trainops=trainop, tensorboard_dir='/tmp/tflearn')
model.fit(feed_dict={input_placeholder: X, target_placeholder: Y})
```

Trainer, Evaluator, and Predictor

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

Code

```
model = Trainer(trainops=[trainop1, trainop2])
model.fit(feed_dict=[{in1: X1, label1: Y1}, {in2: X2, in3: X3, label2:
Y2}])
```

Trainer, Evaluator, and Predictor

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

Code

```
model = Evaluator(network)
model.predict(feed_dict={input_placeholder: X})
```

Trainer, Evaluator, and Predictor

To handle networks that have layer with different behaviors at training and testing time such as dropout and batch normalization:

- Trainer class uses a Boolean variable (**is_training**), that specifies if the network is used for training or testing or predicting. This variable is stored under **tf.GraphKeys.IS_TRAINING** collection, as its first element. So, while defining such layers, this variable should be used as the operational condition:

Code

```
# 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.
```

What Is Keras?

A high-level neural network API, written in Python

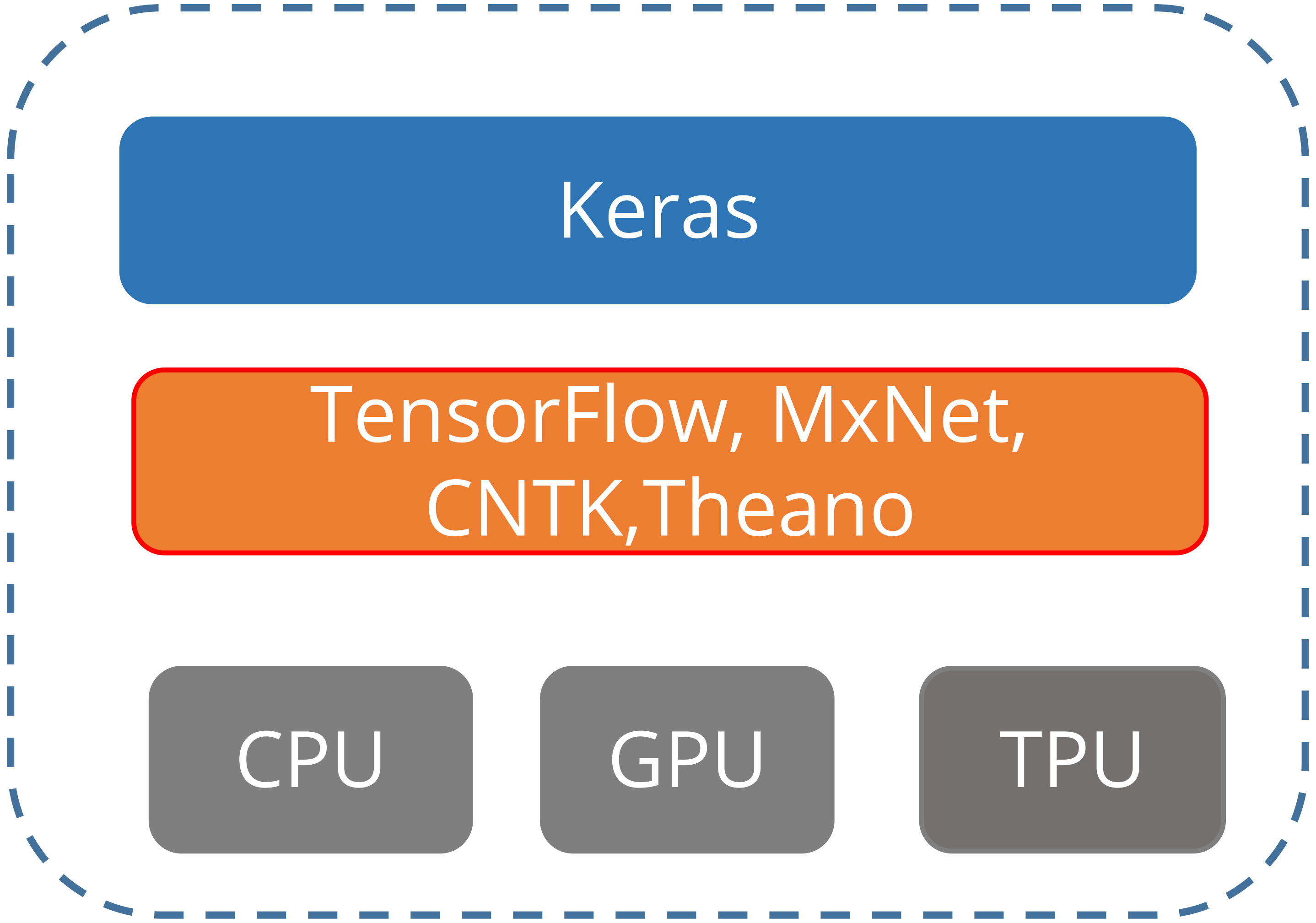
Most powerful and easy to use for developing and evaluating deep learning models

Keras

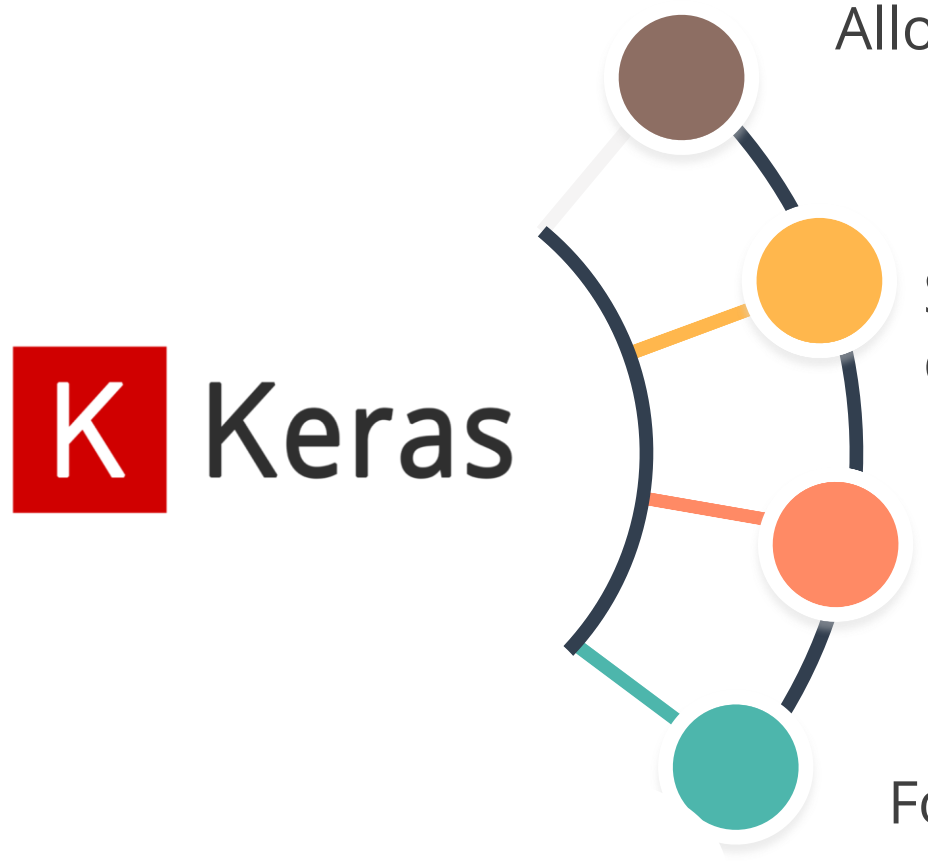
Runs seamlessly on CPU and GPU

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 combination of both

Provides clear and actionable feedback for user error

Follows best practices for reducing cognitive load

Installation of Keras

Installation of Keras is done as follows:

- Install Keras in virtualenv:
  - pip3 install keras
- Install Keras from the GitHub source:
  - Clone Keras using git:
    - git clone https://github.com/keras-team/keras.git
  - cd to the Keras folder and run the install command:
    - cd keras
    - sudo python setup.py install

Creating a Keras Model

1. **Architecture Definition**: Number of layers, number of nodes in layers, and activation function to be used
2. **Compile**: Defines the loss function and details about how optimization works
3. **Fit**: Finalizes the model through back propagation and optimization of weights with input data
4. **Predict:** Predicts with the model prepared

Create the Model

The sequential model is a linear stack of layers.

Code

```
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'))
```

Compile the Model

Code

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

- 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 classification problem.

Fit the Model

Code

```
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

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

Evaluate the Model

Code

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

- Assesses the modeled data set

Predict

Code

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

- Generates prediction on new data

Other Deep Learning Tool: PyTorch

What Is PyTorch?

A deep learning research platform that provides maximum flexibility and speed

A replacement for NumPy to use the power of GPUs

PyTorch

A product of Facebook's artificial intelligence team

Features of PyTorch

The features of Pytorch is as follows:

- Simple Interface
- Hybrid Frontend
- Distributed Training
- Native ONNX Support
- C++ Frontend
- Cloud Partners

PyTorch Ecosystems

Glow, an ML compiler increases the performance of deep learning platform.

PyTorch Geometric, a library for deep learning for irregular input data.

PyTorch

Skorch, high-level library provides full scikit-learn compatibility.

Torchbearer, a library for advanced visualizations.

Installation of PyTorch

Configurations followed to install PyTorch:

PyTorch Build	[✓ Stable(1.2)]	Preview (Nightly)			
Your OS	Linux	MAC	[✓ Windows]		
Package	[✓ Conda]	Pip	LibTorch	Source	
Language	Python 2.7	Python 3.5	[✓ Python 3.6]	Python 3.7	C++
CUDA	9.2	[✓ 10.0]	None		
Run this command	Conda install pytorch torchvision cudotoolkit = 10.0 –c pytorch				

Deep Learning Model with Keras

Problem Statement: A data set is given of diabetes patients with different health parameters make a deep learning classification model to predict.

Access:

- Click on the Labs tab on the left side panel of the LMS. Copy the username and password.
- Click on the Launch Lab button. On the page that appears, enter the username and password, and click Login.

ASSISTED PRACTICE

Loading Dataset

Processing the Data

Define the Model

Compile the Model

Fit the Model

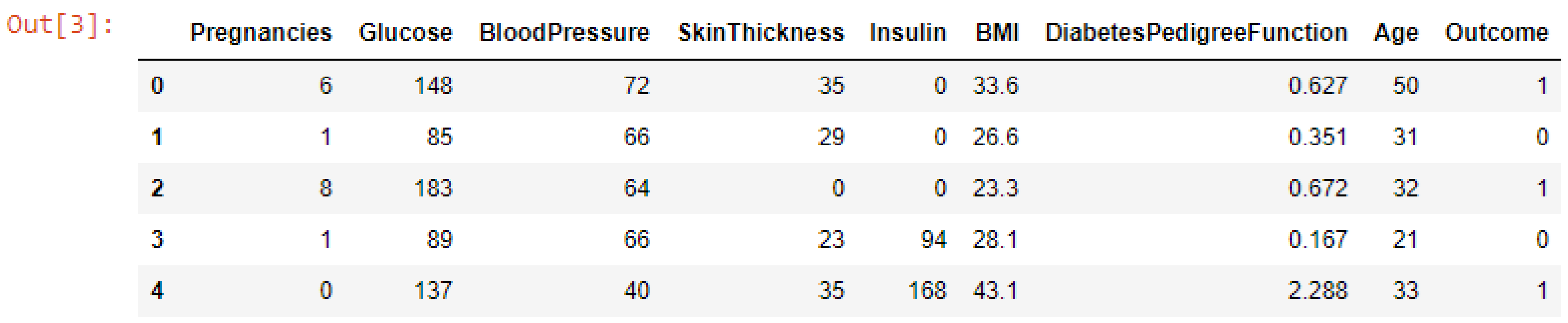
Evaluate the Model

Predict

The dataset is read first by using pandas library and the first five line of the dataset is printed.

Code

```
import pandas as pd
dataset = pd.read_csv('diabetes.csv')
Dataset.head()
```



Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Splitting Dataset

Processing the Data

Define the Model

Compile the Model

Fit the Model

Evaluate the Model

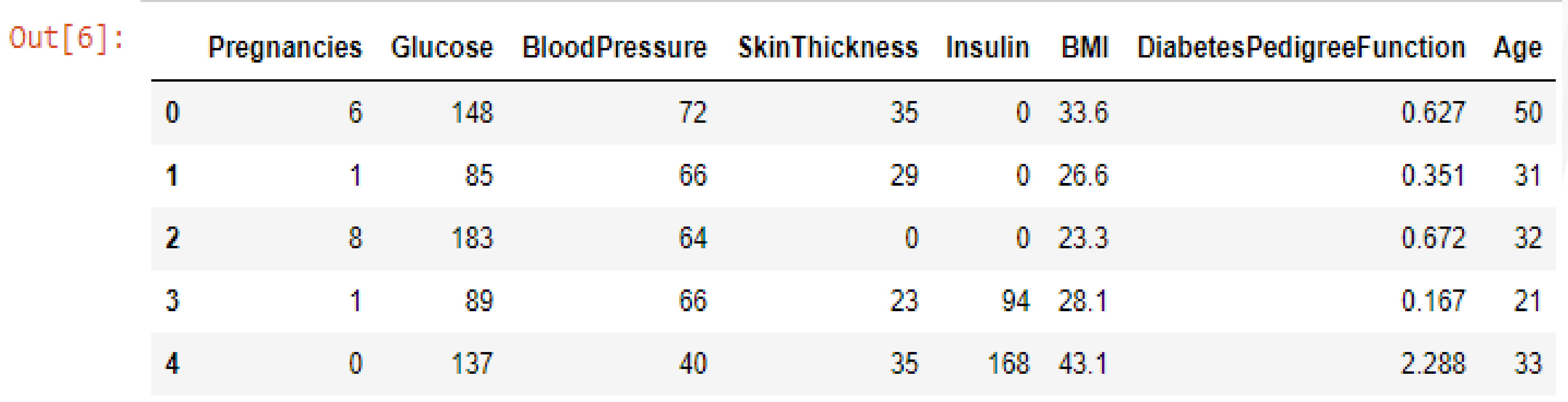
Predict

The dataset is split into input and output.

Code

```
X = dataset.iloc[:,0:8]
y = dataset.iloc[:, 8]

X.head()
```



Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

Splitting Dataset

Processing the Data

Define the Model

Compile the Model

Fit the Model

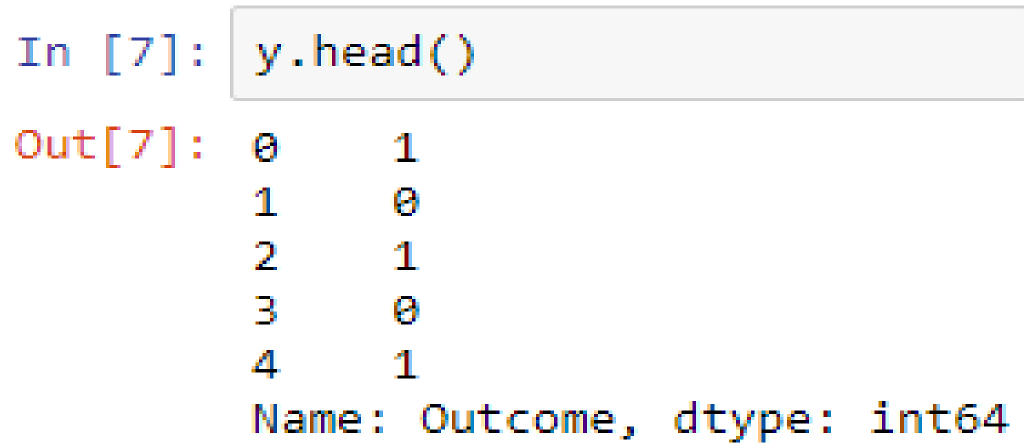
Evaluate the Model

Predict

The dataset is split into input and output.

Code

```
Y.head()
```



```
In [7]: y.head()

Out[7]: 0    1
        1    0
        2    1
        3    0
        4    1
        Name: Outcome, dtype: int64
```

Importing Library

- Processing the Data
- **Define the Model**
- Compile the Model
- Fit the Model
- Evaluate the Model
- Predict

The model is sequential and the layers are defined with Dense class.

Code

```
from keras.models import Sequential
from keras.layers import Dense
```

Creating the Layers

- Processing the Data
- **Define the Model**
- Compile the Model
- Fit the Model
- Evaluate the Model
- Predict

Layer structure of the model:

- The model expects rows of data with eight variables (the input_dim=8 argument).
- The first hidden layer has 12 nodes and uses the ReLU activation function.
- The second hidden layer has 8 nodes and uses the ReLU activation function.
- The output layer has one node and uses the sigmoid activation function.

Code

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Compile the Model

- Processing the Data
- Define the Model
- Compile the Model
- Fit the Model
- Evaluate the Model
- Predict

- Binary Cross Entropy is set as loss function for this classification model
- The optimizer is Adam algorithm
- The metrics is set to accuracy

Code

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

Fit the Model

- Processing the Data
- Define the Model
- Compile the Model
- **Fit the Model**
- Evaluate the Model
- Predict

Training occurs over epochs and each epoch is split into batches.

Code

```
model.fit(X, y, epochs=60, batch_size=10)
```

```
Epoch 1/60
768/768 [==============================] - 0s 345us/step - loss: 4.5930 - acc: 0.4883
Epoch 2/60
768/768 [==============================] - 0s 86us/step - loss: 1.2761 - acc: 0.5326

Epoch 25/60
768/768 [==============================] - 0s 88us/step - loss: 0.5949 - acc: 0.7148
Epoch 26/60
768/768 [==============================] - 0s 88us/step - loss: 0.6145 - acc: 0.6875

Epoch 59/60
768/768 [==============================] - 0s 86us/step - loss: 0.5336 - acc: 0.7344
Epoch 60/60
768/768 [==============================] - 0s 86us/step - loss: 0.5171 - acc: 0.7474
```

Evaluate the Model

Processing the Data

Define the Model

Compile the Model

Fit the Model

Evaluate the Model

Predict

The priority of this demo is only the second part of the evaluate() function, i.e, accuracy.

Code

```
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

```
768/768 [==============================] - 0s 44us/step

Accuracy: 73.31
```

Predict

- Processing the Data
- Define the Model
- Compile the Model
- Fit the Model
- Evaluate the Model
- **Predict**

The predict_classess() generate class predictions for the input.

Code

```
predictions = model.predict_classes(X)
predictions[0:5]
```

```
Out[16]: array([[1],
               [0],
               [1],
               [0],
               [1]], dtype=int32)
```

```
dataset['Outcome'].head()
```

```
Out[17]: 0    1
         1    0
         2    1
         3    0
         4    1
         Name: Outcome, dtype: int64
```

Deep Learning Model with TensorFlow

Problem Statement: Create a deep learning model with MNIST dataset to predict the handwritten digits.

Access:

- Click on the Labs tab on the left side of the LMS panel. Copy the username and password.
- Click on the Launch Lab button. On the page that appears, enter the username and password, and click Login.

Loading the Dataset

Processing the Data

Define the Model

Compile the Model

Fit the Model

Evaluate the Model

Predict

Import tensorflow, and load the MNIST, dataset of handwritten digits, 0 to 9 of image with dimension 28x28

Code

```
import tensorflow as tf

mnist_data = tf.keras.datasets.mnist
x_train, y_train),(x_test, y_test) = mnist_data.load_data()
```

simplilearn

Visualizing the Data

Processing the Data

Define the Model

Compile the Model

Fit the Model

Evaluate the Model

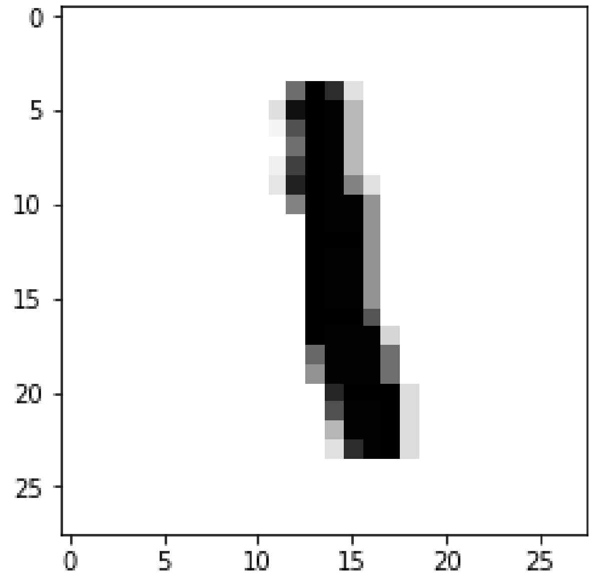
Predict

The digit at the position 6 is printed using matplotlib library.

Code

```
import matplotlib.pyplot as plt

plt.imshow(x_train[6],cmap=plt.cm.binary)
plt.show()
```



```
print(y_train[6])
```

```
1
```

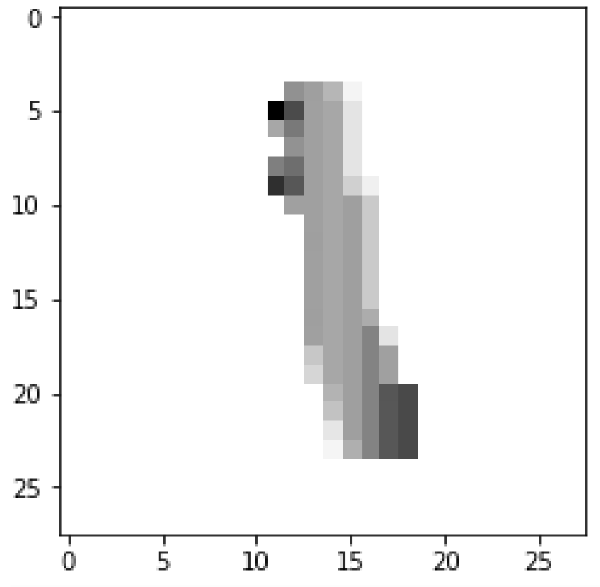
Normalizing the Data

- Processing the Data
- Define the Model
- Compile the Model
- Fit the Model
- Evaluate the Model
- Predict

Data normalization is achieved by tensorflow.keras.utils.normalize() function, and the pixel of the images is normalized from the range 0 to 255 to the range 0 to 1.

Code

```
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
```



- The difference can be seen between original digit and normalized digit.

Define the Model

Processing the Data

Define the Model

Compile the Model

Fit the Model

Evaluate the Model

Predict

A feed forward sequential model is defined:

- Flattening the input layer from 28*28 to 1*128
- Applying two densely connected layer with rectified linear as activation function
- In the final layer, there are 10 nodes, one node for each digit. The activation function is softmax, perfect when desired output is probability distribution of the event over ‘n’ different events

Code

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))

model.add(tf.keras.layers.Dense(10,activation=tf.nn.softmax))
```

Compile the Model

- Processing the Data
- Define the Model
- **Compile the Model**
- Fit the Model
- Evaluate the Model
- Predict

- Sparse Categorical Cross Entropy is set as loss function for this classification model.
- The optimizer is Adam algorithm, straightforward to implement, and gives efficient result.
- The metrics is set to accuracy.

Code

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Fit the Model

- Processing the Data
- Define the Model
- Compile the Model
- **Fit the Model**
- Evaluate the Model
- Predict

Training occurs over epochs and each epoch is split into batches.

Code

```
model.fit(x_train, y_train, epochs=3)
```

```
Epoch 1/3
60000/60000 [==============================] - 4s 60us/sample - loss: 0.2668 - acc: 0.9215
Epoch 2/3
60000/60000 [==============================] - 3s 58us/sample - loss: 0.1103 - acc: 0.9663
Epoch 3/3
60000/60000 [==============================] - 3s 58us/sample - loss: 0.0756 - acc: 0.9764
```

Evaluate the Model

Processing the Data

Define the Model

Compile the Model

Fit the Model

Evaluate the Model

Predict

The evaluate() function returns a list with two values. The first is the loss of the model on the data set, and the second is the accuracy of the model on the dataset.

Code

```
loss, accuracy = model.evaluate(x_test, y_test)
print(loss)
print(accuracy)
```

```
10000/10000 [==============================] - 0s 23us/sample - loss: 0.0934 - acc: 0.9726
0.09337367223678157
0.9726
```

Predict

- Processing the Data
- Define the Model
- Compile the Model
- Fit the Model
- Evaluate the Model
- Predict

The predict() function predicts the digit for the input.

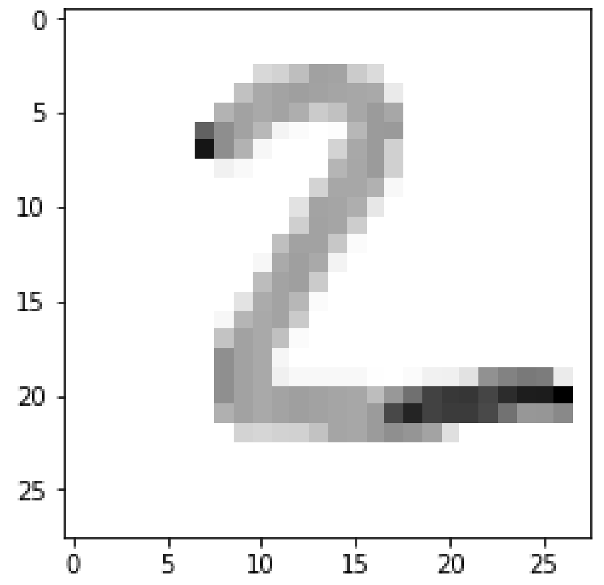
Code

```
predictions = model.predict(x_test)

import numpy as np
print(np.argmax(predictions[1]))
```

```
2
```

```
plt.imshow(x_test[1],cmap=plt.cm.binary)
plt.show()
```



Deep Learning Model with Keras

Problem Statement: Build a deep learning model using Fashion-MNIST, dataset of fashion articles with 10 classes and each image is 28*28 pixel.

Access:

- Click on the Labs tab on the left side of the LMS panel. Copy the username and password.
- Click on the Launch Lab button. On the page that appears, enter the username and password, and click Login.

ASSISTED PRACTICE

Importing the Library

Processing the Data

Building the Network

Define Loss Function and Optimizer

Train the Network

Evaluate the Model

Code

```
import torch
import torchvision
import torchvision.transforms as transforms
```

Creating Image Normalizer

Processing the Data

Building the Network

Define Loss Function and Optimizer

Train the Network

Evaluate the Model

An object is created using Compose() function to normalize the image data.

Code

```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,),
(0.5,))])
```

Creating Data Loader

- Processing the Data
- Building the Network
- Define Loss Function and Optimizer
- Train the Network
- Evaluate the Model

Creating data loader object for training data:

Code

```
traindata = torchvision.datasets.FashionMNIST(root='./data', 
train=True,
           download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(traindata, 
batch_size=4,
              shuffle=True)
```

Creating Data Loader

Processing the Data

Building the Network

Define Loss Function and Optimizer

Train the Network

Evaluate the Model

Creating data loader object for testing data:

Code

```
testdata = torchvision.datasets.FashionMNIST(root='./data',
train=False,
          download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testdata,
batch_size=4,
             shuffle=False)
```

Visualizing the Image Data

Processing the Data

Building the Network

Define Loss Function and Optimizer

Train the Network

Evaluate the Model

Code

```
import matplotlib.pyplot as plt
import numpy as np


def show_image(image):
    image = image / 2 + 0.5 # unnormalize
    np_img = image.numpy()
    plt.imshow(np.transpose(np_img, (1, 2, 0)))
```

Visualizing the Image Data

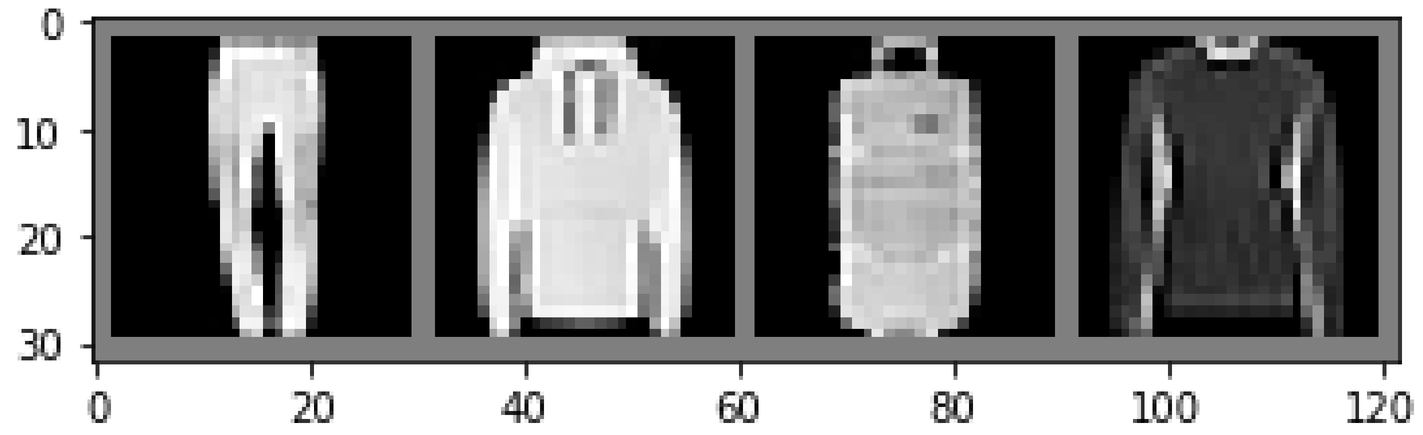
- Processing the Data
- Building the Network
- Define Loss Function and Optimizer
- Train the Network
- Evaluate the Model

Code

```
dataiter = iter(trainloader)
images, labels = dataiter.next()

show_image(torchvision.utils.make_grid(images))
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

```
Trouser Pullover   Bag Pullover
```



Building the Network

Processing the Data

Building the Network

Define Loss Function and Optimizer

Train the Network

Evaluate the Model

Code

```
import torch.nn as nn
import torch.nn.functional as F

class Network(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(x.shape[0], -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.softmax(x, dim=1)

        return x
```

Define Loss Function and Optimizer

- Cross entropy is set as loss function for this classification model.
- The optimizer is Adam algorithm.

Code

```
from torch import optim

model = Network()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.1)
```

Train the Network

Processing the Data

Building the Network

Define Loss Function and Optimizer

Train the Network

Evaluate the Model

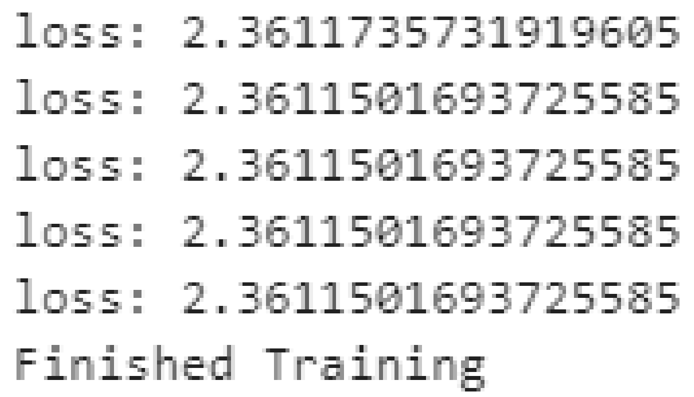
Code

```
for epoch in range(5):
    running_loss = 0.0
    for inputs, labels in trainloader:
        output = model(inputs)
        loss = criterion(output, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    else:
        print(f"loss: {running_loss/len(trainloader)}")

print('Finished Training')
```



```
loss: 2.3611735731919605
loss: 2.3611501693725585
loss: 2.3611501693725585
loss: 2.3611501693725585
loss: 2.3611501693725585
Finished Training
```

Train the Network

- Processing the Data
- Building the Network
- Define Loss Function and Optimizer
- Train the Network
- Evaluate the Model

Code

```
correct = 0
total = 0

with torch.no_grad():
    for data in testloader:
    images, labels = data
    outputs = model(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d
%%' % (
100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 10 %
```

Deep Learning Model with Caffe2

Problem Statement: Make a deep learning model with MNIST data using Caffe2.

Access:

- Click on the Labs tab on the left side of the LMS panel. Copy the username and password.

- Click on the Launch Lab button. On the page that appears, enter the username and password, and click Login.

ASSISTED PRACTICE

Loading the Data

Loading the Data

Constructing the Model

Training the Model

Testing the Model

Deploying the Model

Running the Training

Code

```
from keras.datasets.mnist import load_data

(trainX, trainy), (testX, testy) = load_data()

print('Train', trainX.shape, trainy.shape)
print('Test', testX.shape, testy.shape)
```

Importing Library

Importing the Library

Loading the Data

Constructing the Model

Training the Model

Testing the Model

Deploying the Model

Running the Training

Code

```
from caffe2.python import (
    brew,
    core,
    model_helper,
    net_drawer,
    optimizer,
    visualize,
    workspace,
)

core.GlobalInit(['caffe2', '--caffe2_log_level=0'])
print("Necessities imported!")

USE_LENET_MODEL = True
```

Loading the Data

Importing the Library

Loading the Data

Constructing the Model

Training the Model

Testing the Model

Deploying the Model

Running the Training

Code

```
def DownloadResource(url, path):
    '''Downloads resources from s3 by url and unzips them to
the provided path'''
    import requests, zipfile, StringIO
    print("Downloading... {} to {}".format(url, path))
    r = requests.get(url, stream=True)
    z = zipfile.ZipFile(StringIO.StringIO(r.content))
    z.extractall(path)
```

Loading the Data

Set up the paths for the necessary directories.

Code

```
current_folder = os.path.join(os.path.expanduser('~'),
'caffe2_notebooks')
data_folder = os.path.join(current_folder, 'tutorial_data',
'mnist')
root_folder = os.path.join(current_folder, 'tutorial_files',
'tutorial_mnist')
db_missing = False
```

Loading the Data

- Importing the Library
- Loading the Data
- Constructing the Model
- Training the Model
- Testing the Model
- Deploying the Model
- Running the Training

Check if the data folder already exists.

Code

```
if os.path.exists(os.path.join(data_folder,"mnist-train-
nchw-lmdb")):
    print("lmdb train db found!")
else:
    db_missing = True
```

Loading the Data

Check if the testing LMDB exists in the data folder.

Code

```
current_folder = os.path.join(os.path.expanduser('~'),
'caffe2_notebooks')
data_folder = os.path.join(current_folder, 'tutorial_data',
'mnist')
root_folder = os.path.join(current_folder, 'tutorial_files',
'tutorial_mnist')
db_missing = False
```

Loading the Data

Attempt to download the database if it is missing from either of the folders.

Code

```
if db_missing:
    print("one or both of the MNIST lmbd dbs not found!!")
    db_url = "http://download.caffe2.ai/databases/mnist-
lmdb.zip"
    try:
        DownloadResource(db_url, data_folder)
    except Exception as ex:
        print("Failed to download dataset. Please download
it manually from {}".format(db_url))
        print("Unzip it and place the two database folders
here: {}".format(data_folder))
        raise ex
```

Loading the Data

Clean up the statistics from any of the old runs.

Code

```
if os.path.exists(root_folder):
    print("Looks like you ran this before, so we need to
cleanup those old files...")
    shutil.rmtree(root_folder)

os.makedirs(root_folder)
workspace.ResetWorkspace(root_folder)
```

Loading the Data

Importing the Library

Loading the Data

Constructing the Model

Training the Model

Testing the Model

Deploying the Model

Running the Training

Code

```
if os.path.exists(root_folder):
    print("Looks like you ran this before, so we need to
cleanup those old files...")
    shutil.rmtree(root_folder)

os.makedirs(root_folder)
workspace.ResetWorkspace(root_folder)
```

Loading the Data

For the sake of modularity, we will separate the construction of the model into different parts:

- The data input part (AddInput function)
- The main computation part (AddModel function)
- The training part is where gradient operators, optimization algorithm, etc. are added (AddTrainingOperators function)
- The bookkeeping part where you just print the statistics for inspection (AddBookkeepingOperators function)

Constructing the Model

- Importing the Library
- Loading the Data
- Constructing the Model
- Training the Model
- Testing the Model
- Deploying the Model
- Running the Training

AddInput function loads the data from a database.

Code

```
def AddInput(model, batch_size, db, db_type):

    data_uint8, label = model.TensorProtosDBInput(
        [], ["data_uint8", "label"], batch_size=batch_size,
        db=db, db_type=db_type)
    data = model.Cast(data_uint8, "data",
to=core.DataType.FLOAT)
    data = model.Scale(data, data, scale=float(1./256))
    data = model.StopGradient(data, data)
    return data, label
```

Constructing the Model

When the flag USE_LENET_MODEL is false, MLP model definition is used.

Code

```
def AddMLPModel(model, data):
    size = 28 * 28 * 1
    sizes = [size, size * 2, size * 2, 10]
    layer = data
    for i in range(len(sizes) - 1):
        layer = brew.fc(model, layer, 'dense_{}'.format(i),
dim_in=sizes[i], dim_out=sizes[i + 1])
        layer = brew.relu(model, layer, 'relu_{}'.format(i))
    softmax = brew.softmax(model, layer, 'softmax')
    return softmax
```

Constructing the Model

- Importing the Library
- Loading the Data
- Constructing the Model
- Training the Model
- Testing the Model
- Deploying the Model
- Running the Training

When the flag USE_LENET_MODEL is true, MLP model definition is used.

Code

```
def AddLeNetModel(model, data):

    conv1 = brew.conv(model, data, 'conv1', dim_in=1,
dim_out=20, kernel=5)
    pool1 = brew.max_pool(model, conv1, 'pool1', kernel=2,
stride=2)
    conv2 = brew.conv(model, pool1, 'conv2', dim_in=20,
dim_out=50, kernel=5)
    pool2 = brew.max_pool(model, conv2, 'pool2', kernel=2,
stride=2)
    fc3 = brew.fc(model, pool2, 'fc3', dim_in=50 * 4 * 4,
dim_out=500)
    relu3 = brew.relu(model, fc3, 'relu3')
    pred = brew.fc(model, relu3, 'pred', dim_in=500,
dim_out=10)
    softmax = brew.softmax(model, pred, 'softmax')

    return softmax
```

Constructing the Model

The AddModel function allows you to switch easily from MLP to LeNet model. Change USE_LENET_MODEL at the very top of the notebook and rerun the whole code.

Code

```
def AddModel(model, data):
    if USE_LENET_MODEL:
        return AddLeNetModel(model, data)
    else:
        return AddMLPModel(model, data)
```

Constructing the Model

The AddAccuracy function acts as an accuracy operator to the model. It uses the softmax scores and the input training labels.

Code

```
def AddAccuracy(model, softmax, label):
    accuracy = brew.accuracy(model, [softmax, label],
"accuracy")
    return accuracy
```

Constructing the Model

Importing the Library

Loading the Data

Constructing the Model

Training the Model

Testing the Model

Deploying the Model

Running the Training

Add training operators.

Code

```
def AddTrainingOperators(model, softmax, label):
    xent = model.LabelCrossEntropy([softmax, label], 'xent')
    loss = model.AveragedLoss(xent, "loss")
    AddAccuracy(model, softmax, label)
    model.AddGradientOperators([loss])
    optimizer.build_sgd(
        model,
        base_learning_rate=0.1,
        policy="step",
        stepsize=1,
        gamma=0.999,
    )
```

Constructing the Model

Add bookkeeping operators.

Code

```
def AddBookkeepingOperators(model):
    model.Print('accuracy', [], to_file=1)
    model.Print('loss', [], to_file=1)
    for param in model.params:
        model.Summarize(param, [], to_file=1)
        model.Summarize(model.param_to_grad[param], [],
to_file=1)
```

Training the Model

- Importing the Library
- Loading the Data
- Constructing the Model
- Training the Model
- Testing the Model
- Deploying the Model
- Running the Training

Code

```
arg_scope = {"order": "NCHW"}
train_model = model_helper.ModelHelper(name="mnist_train",
arg_scope=arg_scope)
data, label = AddInput(
    train_model, batch_size=64,
    db=os.path.join(data_folder, 'mnist-train-nchw-lmdb'),
    db_type='lmdb')
softmax = AddModel(train_model, data)
AddTrainingOperators(train_model, softmax, label)
AddBookkeepingOperators(train_model)
```

Testing the Model

- Importing the Library
- Loading the Data
- Constructing the Model
- Training the Model
- Testing the Model
- Deploying the Model
- Running the Training

Code

```
test_model = model_helper.ModelHelper(
    name="mnist_test", arg_scope=arg_scope,
init_params=False)
data, label = AddInput(
    test_model, batch_size=100,
    db=os.path.join(data_folder, 'mnist-test-nchw-lmdb'),
    db_type='lmdb')
softmax = AddModel(test_model, data)
AddAccuracy(test_model, softmax, label)
```

Deploying the Model

- Importing the Library
- Loading the Data
- Constructing the Model
- Training the Model
- Testing the Model
- **Deploying the Model**
- Running the Training

Code

```
deploy_model = model_helper.ModelHelper(
    name="mnist_deploy", arg_scope=arg_scope,
init_params=False)
AddModel(deploy_model, "data")
```

Running the Training

Importing the Library

Loading the Data

Constructing the Model

Training the Model

Testing the Model

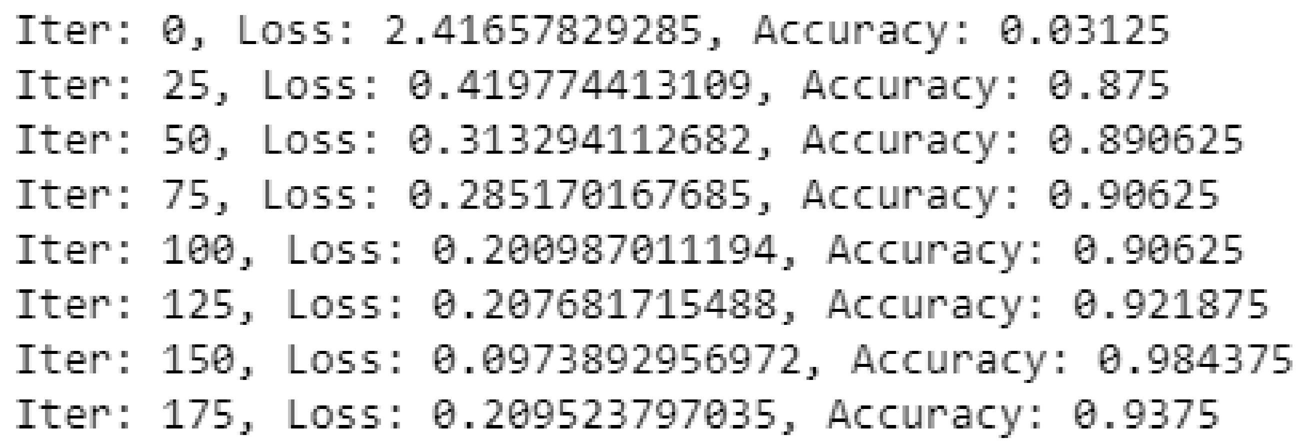
Deploying the Model

Running the Training

Code

```
workspace.RunNetOnce(train_model.param_init_net)
workspace.CreateNet(train_model.net, overwrite=True)
total_iters = 200
accuracy = np.zeros(total_iters)
loss = np.zeros(total_iters)

for i in range(total_iters):
    workspace.RunNet(train_model.net)
    accuracy[i] = workspace.blobs['accuracy']
    loss[i] = workspace.blobs['loss']
    if i % 25 == 0:
        print("Iter: {}, Loss: {}, Accuracy:
{}".format(i,loss[i],accuracy[i]))
```

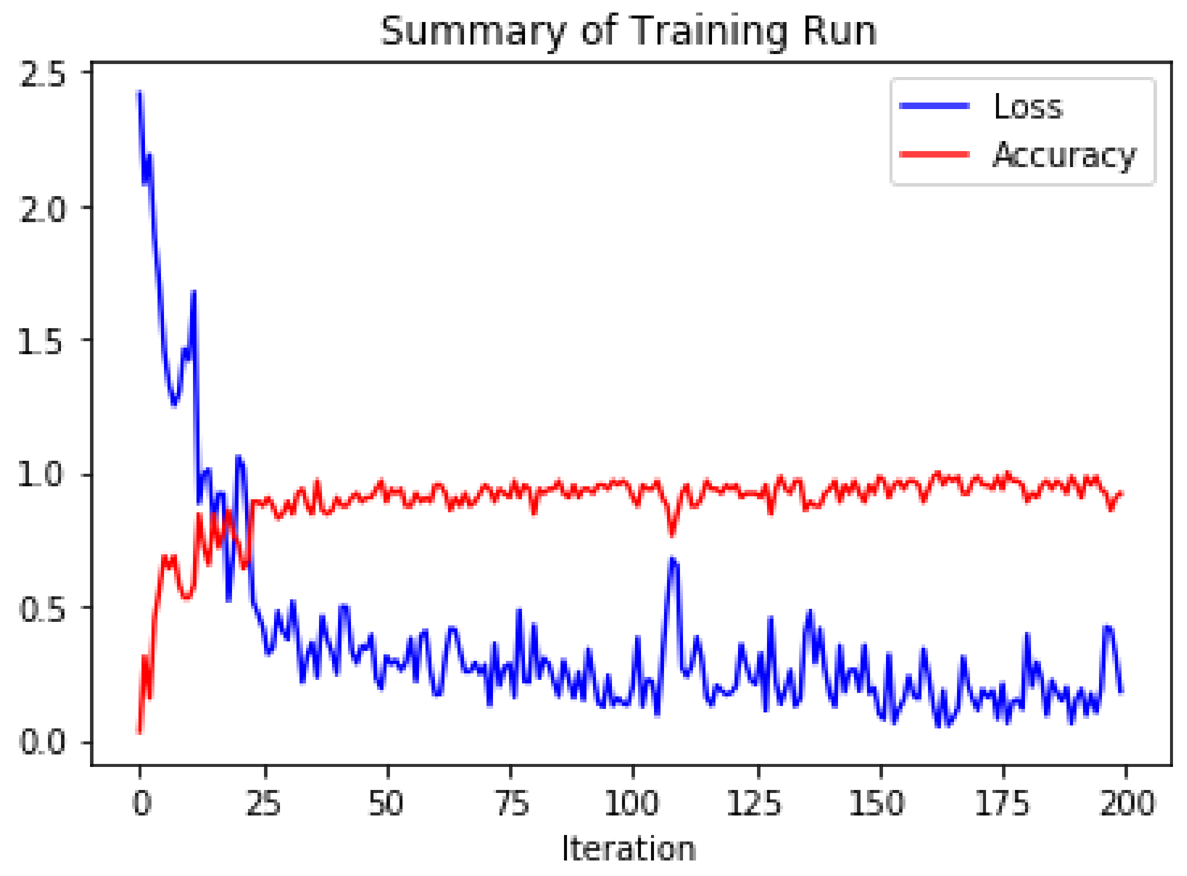


```
Iter: 0, Loss: 2.41657829285, Accuracy: 0.03125
Iter: 25, Loss: 0.419774413109, Accuracy: 0.875
Iter: 50, Loss: 0.313294112682, Accuracy: 0.890625
Iter: 75, Loss: 0.285170167685, Accuracy: 0.90625
Iter: 100, Loss: 0.200987011194, Accuracy: 0.90625
Iter: 125, Loss: 0.207681715488, Accuracy: 0.921875
Iter: 150, Loss: 0.0973892956972, Accuracy: 0.984375
Iter: 175, Loss: 0.209523797035, Accuracy: 0.9375
```

Running the Training

Code

```
pyplot.plot(loss, 'b')
pyplot.plot(accuracy, 'r')
pyplot.title("Summary of Training Run")
pyplot.xlabel("Iteration")
pyplot.legend(('Loss', 'Accuracy'), loc='upper right')
```



Deep Learning Model with Python

Problem Statement: Make a deep learning model with Python.

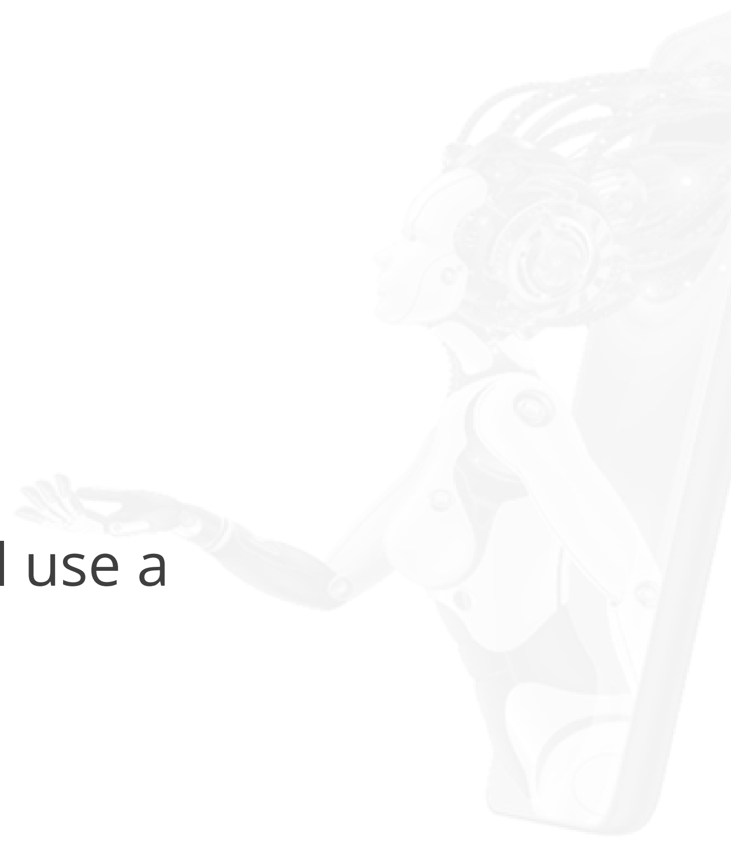
Access:

- Click on the Labs tab on the left side of the LMS panel. Copy the username and password.
- Click on the Launch Lab button. On the page that appears, enter the username and password, and click Login.

Creating the Neural Network Class

Neural network consist of the following components:

- An input layer, x
- An arbitrary amount of hidden layers
- An output layer, ŷ
- A set of weights and biases between each layer, W and b
- A choice of activation function for each hidden layer, σ. In this tutorial, you’ll use a Sigmoid activation function



Creating the Neural Network Class

Code

```
class NeuralNetwork:

    def __init__(self, x, y):
        self.input        = x
        self.weights1     = np.random.rand(self.input.shape[1],4)
        self.weights2     = np.random.rand(4,1)
        self.y            = y
        self.output       = np.zeros(y.shape)
```

Creating the Neural Network Class

Adding backpropagation and loss function in the class NeuralNetwork

Code

```
  def backprop(self):

        # application of the chain rule to find derivative of the loss function with
respect to weights2 and weights1

        d_weights2 = np.dot(self.layer1.T, (2*(self.y - self.output) *
sigmoid_derivative(self.output)))
        d_weights1 = np.dot(self.input.T,  (np.dot(2*(self.y - self.output) *
sigmoid_derivative(self.output), self.weights2.T) * sigmoid_derivative(self.layer1)))

        # update the weights with the derivative (slope) of the loss function

        self.weights1 += d_weights1
        self.weights2 += d_weights2
```

Creating the Neural Network Class

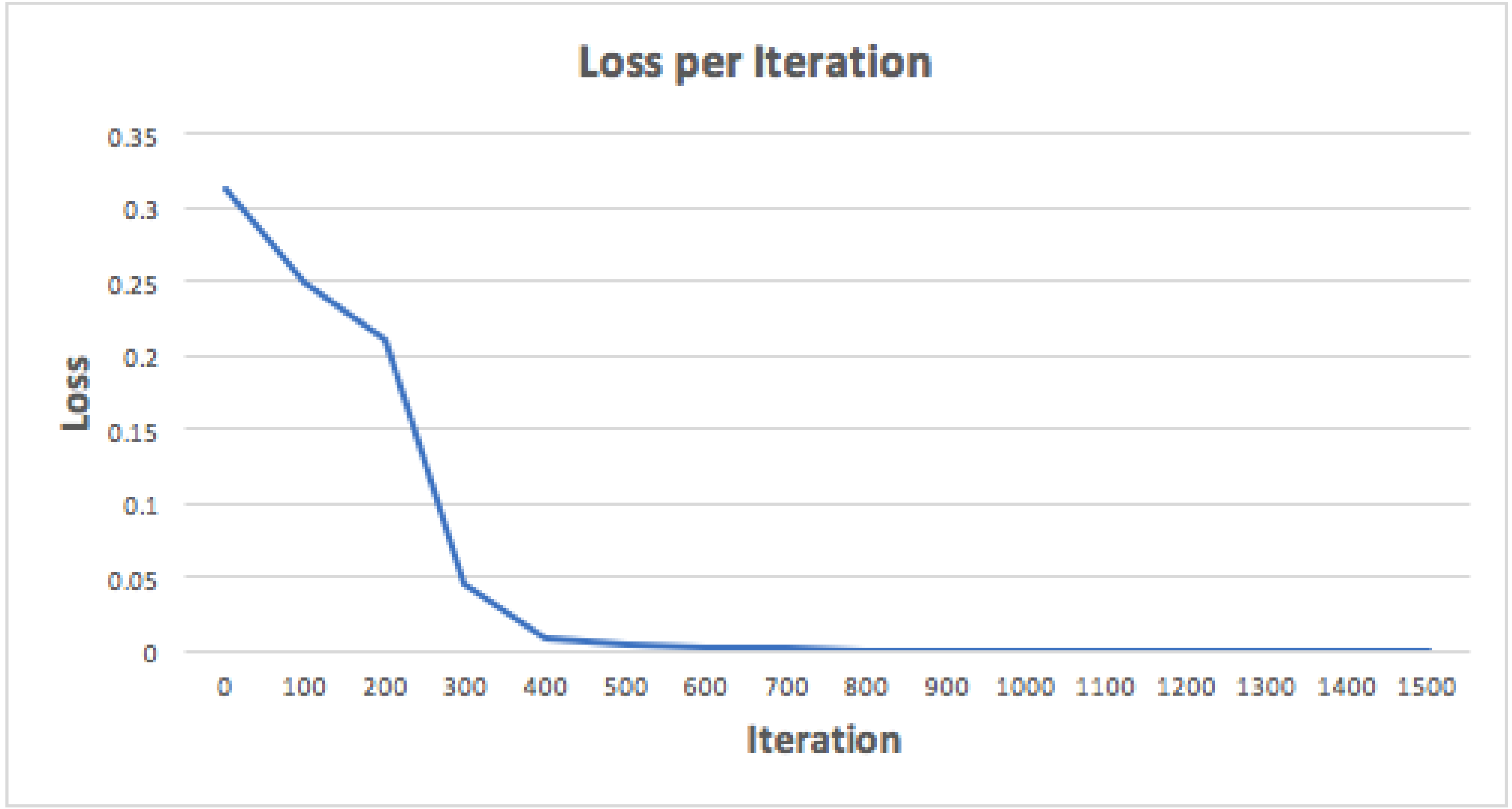
We are creating a Feed Forward function in NeuralNetwork class.

Code

```
def feedforward(self):
       self.layer1 = sigmoid(np.dot(self.input, self.weights1))
       self.output = sigmoid(np.dot(self.layer1, self.weights2)))
```

Output of the Neural Network

The loss of neural network after 1500 iterations:



Output of the Neural Network

Prediction after 1500 iterations:

Prediction	Y (actual)
0.023	0
0.979	1
0.975	1
0.025	0

Key Takeaways

You are now able to:

- Explain and define deep learning models
- Determine suitable loss function for deep learning models
- Apply python to create a deep learning model without any deep learning framework
- Build a deep neural network with Keras, Caffe, PyTorch, and Tensorflow deep learning framework

Knowledge Check

Knowledge Check 1	What is the minimum number of hidden layers a neural network should have to be qualified as a deep neural network?

a. 1

b. 2

c. 3

d. All of the above

Knowledge Check

1

What is the minimum number of hidden layers a neural network should consists to be qualified as deep neural network?

a. 1

b. 2

c. 3

d. All of the above

The correct answer is **b**

If a neural network contains one hidden layer it is called shallow neural network and a shallow neural becomes a deep neural network when one more hidden layer adds up.

Knowledge Check

2

Which of the following deep learning ecosystems does Glow, an ML compiler belong to?

a. PyTorch

b. Keras

c. Tensorflow

d. None of the above

Knowledge Check

2

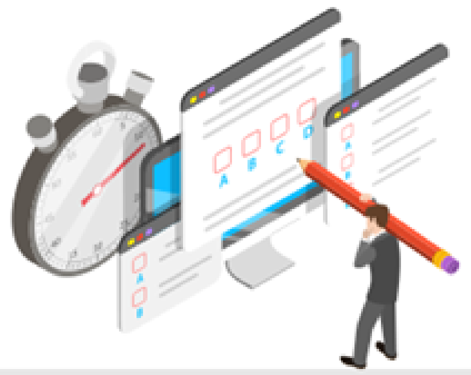
Which of the following deep learning ecosystems does Glow, an ML compiler belong to?

a. PyTorch

b. Keras

c. Tensorflow

d. None of the above



The correct answer is **a**

Glow belongs to PyTorch ecosystem along with Skorch, Torchbearer, and PyTorch Geometric.

Knowledge Check

3

Which of the following loss functions is best suited to build a regression model with outlier?

a. MAE

b. MSE

c. Both MAE and MSE

d. None of the above

Knowledge Check

3

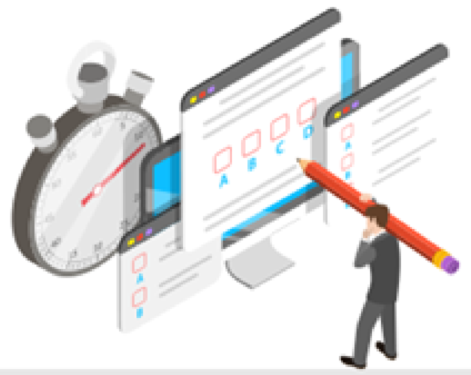
Which of the following loss functions is best suited to build a regression model with outlier?

a. MAE

b. MSE

c. Both MAE and MSE

d. None of the above



The correct answer is **a**

MAE is the suitable loss function to build a regression model with outlier while for a dataset without outlier it is MSE.

Knowledge Check

4

Which of the following deep learning frameworks uses Tensorflow backend?

a. Keras

b. PyTorch

c. Caffe

d. None of the above

Knowledge Check 4

Which of the following deep learning frameworks uses Tensorflow backend?s

a. Keras

b. PyTorch

c. Caffe

d. None of the above

The correct answer is **a**

Keras uses Tensorflow backend along with Theano and Mxnet.

Chars74k Image Classification

Problem Statement: Build a deep learning convolutional neural network to recognize characters using Chars74k dataset.

Objective: Build a neural network-based classification model to recognize characters using the following metrics:

Use 4 convolution layers with 3*3 kernel and activation function as ReLU. Add maximum pooling layers after every other convolution layer and 2 hidden layers with dropout.

Access: Click on the Labs tab on the left side of the LMS panel. Copy or note the username and password that are generated. Click on the Launch Lab button. On the page that appears, enter the username and password in the respective fields, and click Login.

Thank You