UBDA Workshop:
State-of-the-art in Deep Learning Frameworks

University Research Facility in Big Data Analytics

2020 April

UBDA website:
https://ubda.polyu.edu.hk/

The Hong Kong Polytechnic University
香港理工大學

Opening Minds • Shaping the Future
啟迪思維 • 成就未來

Dillian Wong [UBDA]

UBDA Workshop: State-of-the-art in Deep Learning Frameworks
Ice-Breaker Activity

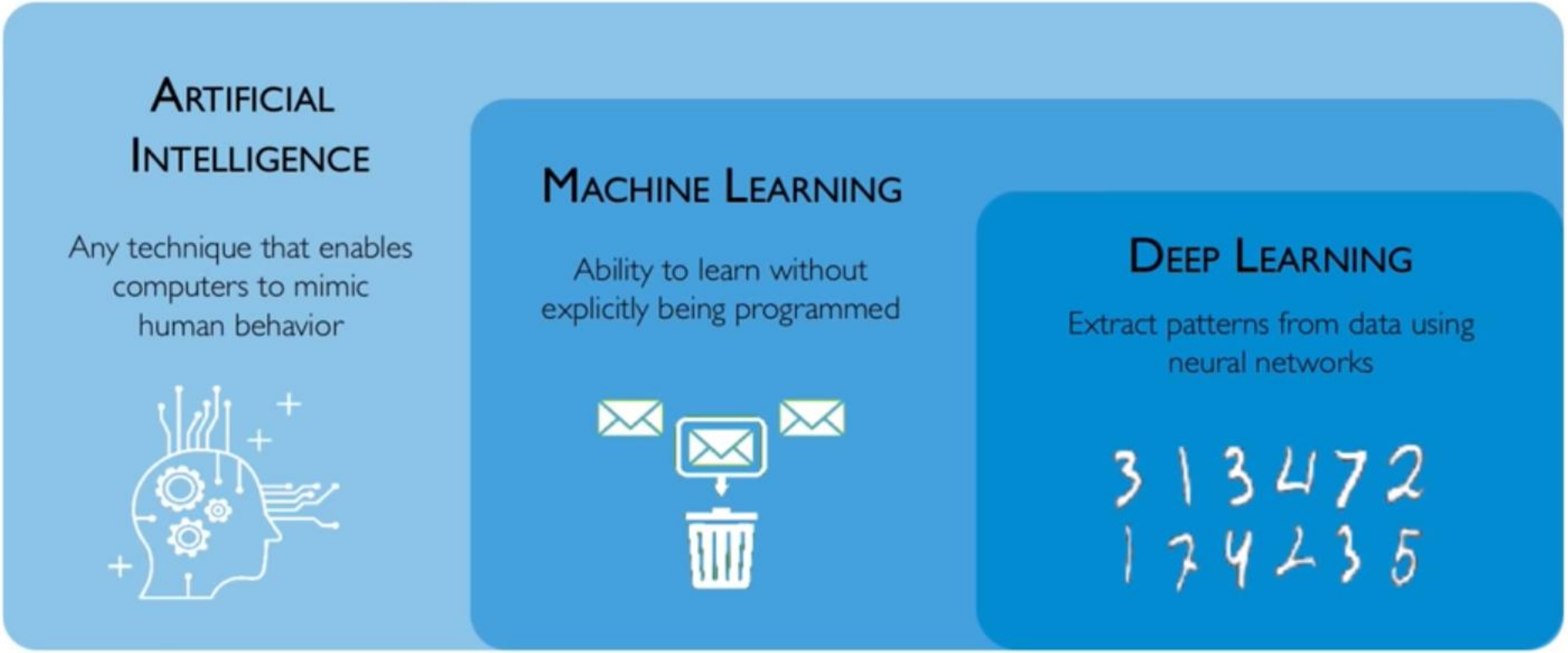
http://etc.ch/WEwL

Which Framework have you used in your research?

Which Framework do you plan to use in your research?

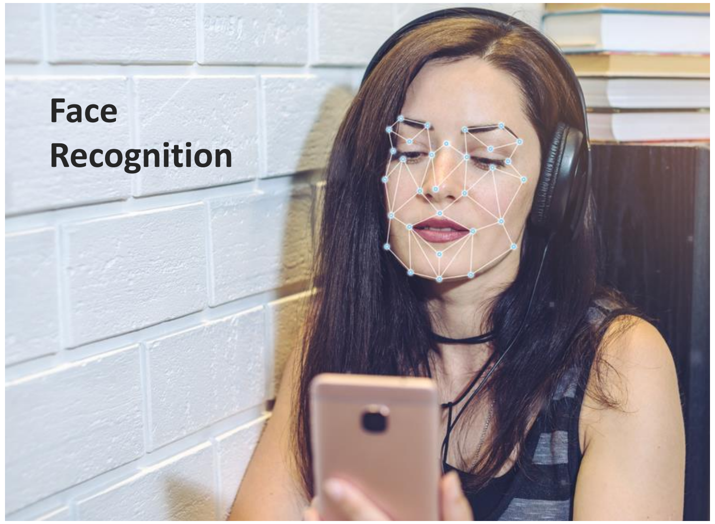
Let's poll now!

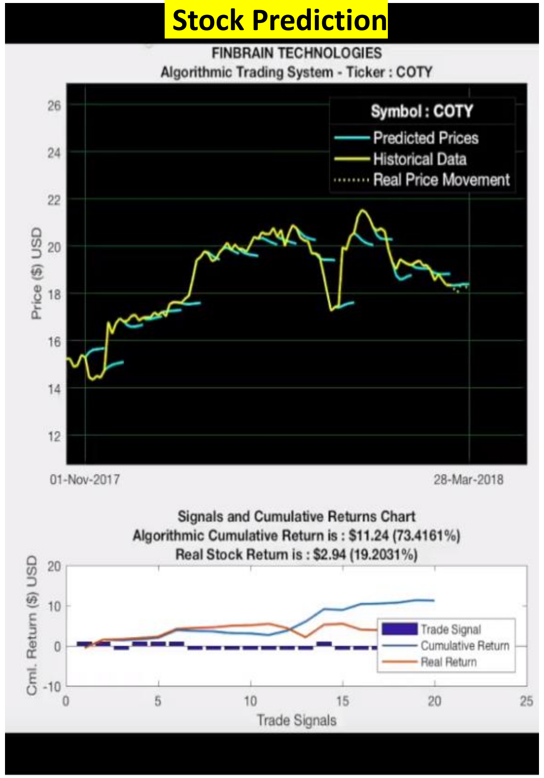
What is Deep Learning?

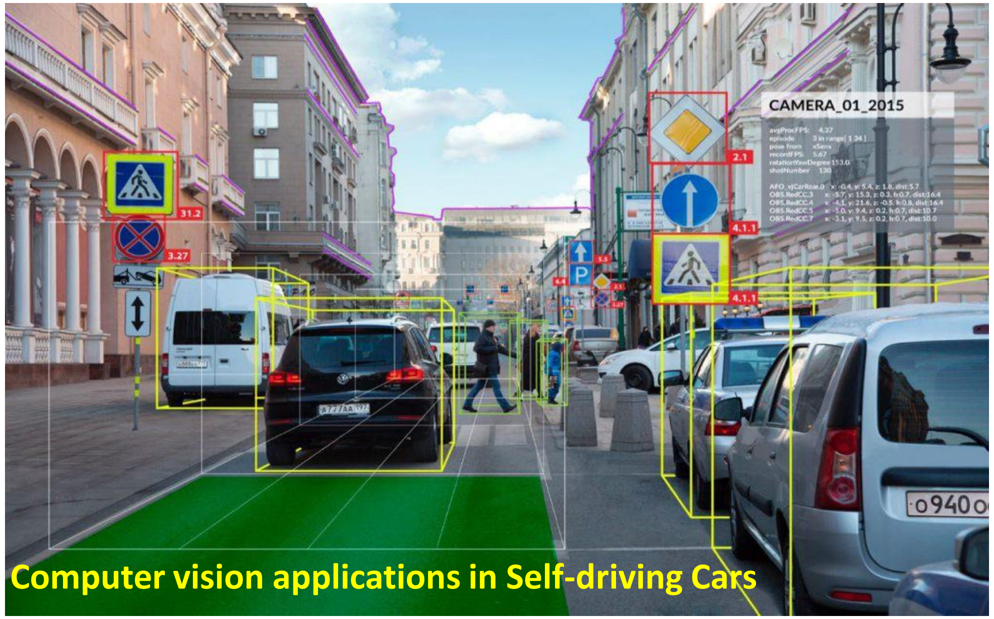


Source: http://www.introtodeeplearning.com

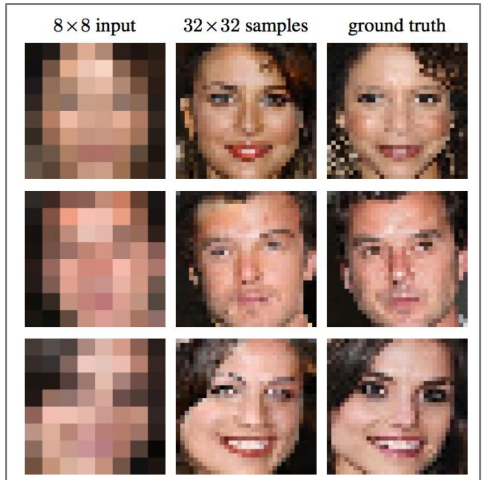
Deep Learning Daily Life Examples







Computer vision applications in Self-driving Cars



Pixel Restoration


Chatbot with Deep learning

Deep Learning Tools available in 2020

SWIFT

Chainer

DL4J

Keras

GLUON

Sonnet

ONNX

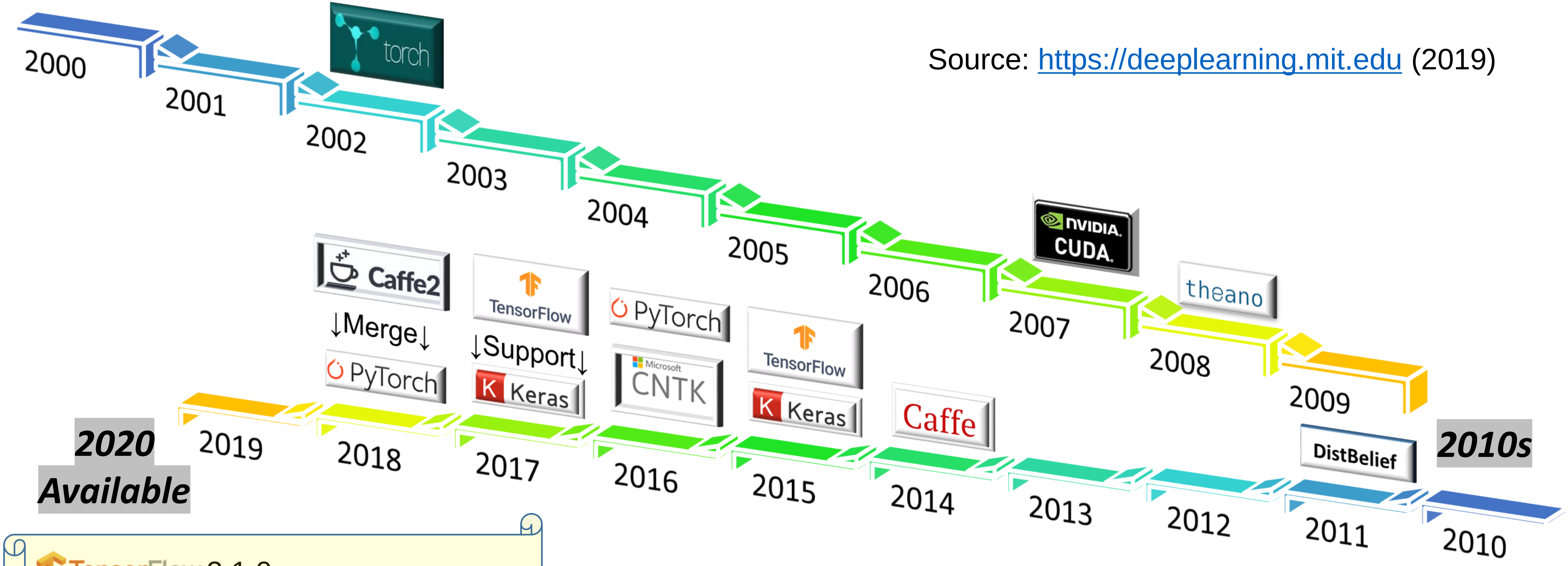
TensorFlow

PYTORCH

mxnet

History of Deep Learning Tools since 2000

Source: https://deeplearning.mit.edu (2019)



TensorFlow 2.1.0
Keras 2.3.1
PyTorch 1.4.0

Content of this workshop

Most Popular Deep Learning Framework

Keras

TensorFlow

PyTorch

How UBDA supports your Deep Learning work?

Technical Support & Research Support

Deep Learning Framework

Keras vs TensorFlow vs PyTorch

Source: https://www.mygreatlearning.com/blog/computer-vision-using-pytorch/ (2020)

	Keras	TensorFlow	PyTorch
Level of API	High-level API	Both high & low level APIs	Lower-level API
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs

Deep Learning Framework

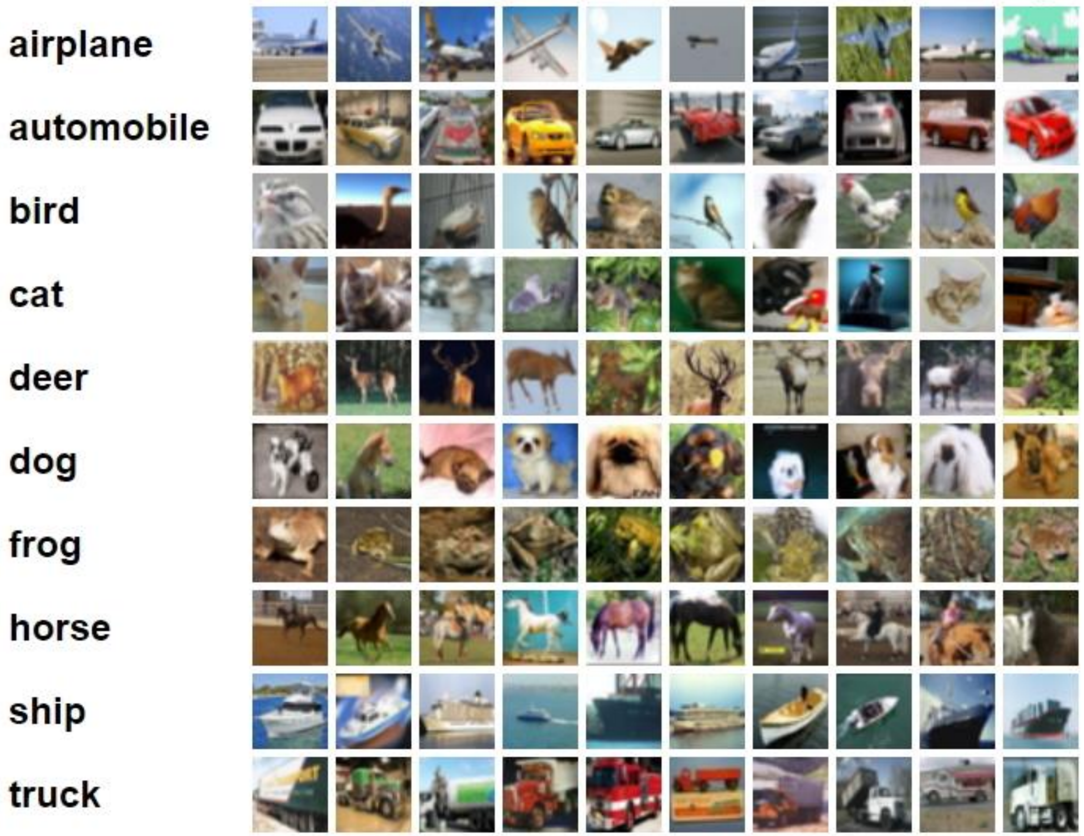
Architecture of PyTorch

Package	Description
torch	The top-level PyTorch package and tensor library
torch.nn	A subpackage that contains modules and extensible classes for building neural networks.
torch.autograd	A subpackage that supports all the differentiable Tensor operations in PyTorch
torch.nn.functional	A functional interface that contains typical operations used for building neural networks like loss functions, activation functions, and convolution operations.
torch.optim	A subpackage that contains standard optimization operations like SGD and Adam
torch.utils	A subpackage that contains utility classes like data sets and data loaders that make data preprocessing easier
torchvision	A package that provides access to popular datasets, model architectures, and image transformations for computer vision.

- PyTorch is made of different modules which help in executing deep learning models for CV and Natural Language Processing (NLP)

Deep Learning Framework

Code Study 1: Train an image classifier in PyTorch



- Setting up the Environment

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

UBDA
UNIVERSITY RESEARCH FACILITY IN BIG DATA ANALYTICS
大數據分析中心實驗室
THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

Deep Learning Framework

Code Study 1: Train an image classifier in PyTorch

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

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

Out:

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

- Transform torchvision datasets to Tensors of normalized range

Deep Learning Framework

Code Study 1: Train an image classifier in PyTorch

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


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

- Define a Convolutional Neural Network

Deep Learning Framework

Code Study 1: Train an image classifier in PyTorch

- Define Loss function and optimizer

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Deep Learning Framework

Code Study 1: Train an image classifier in PyTorch

- Train the network

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

Out:

```
[1,  2000] loss: 2.197
[1,  4000] loss: 1.871
[1,  6000] loss: 1.670
[1,  8000] loss: 1.582
[1, 10000] loss: 1.508
[1, 12000] loss: 1.464
[2,  2000] loss: 1.390
[2,  4000] loss: 1.390
[2,  6000] loss: 1.327
[2,  8000] loss: 1.331
[2, 10000] loss: 1.293
[2, 12000] loss: 1.292
Finished Training
```

Deep Learning Framework
Code Study 1: Train an image classifier in PyTorch

```python
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(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))
```

Out:

```
Accuracy of the network on the 10000 test images: 55 %
```

- Test the network on the test data (overall)

Deep Learning Framework

Code Study 1: Train an image classifier in PyTorch

- Test the network on the test data (in class)

```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

Out:

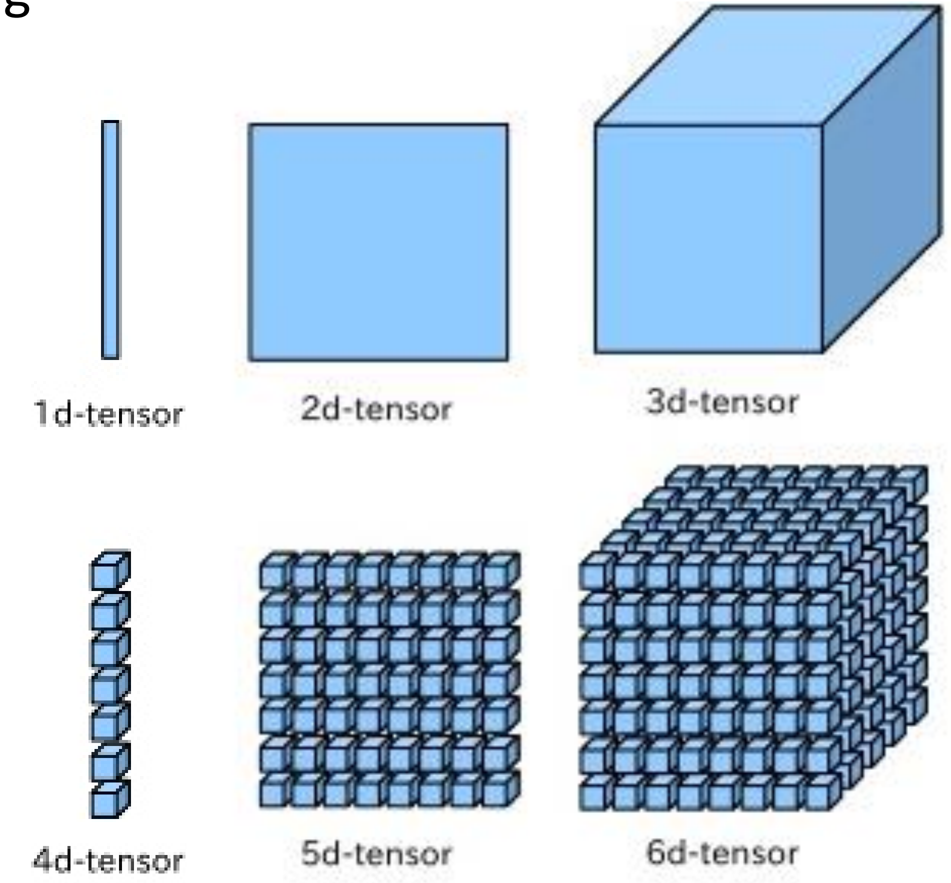
```
Accuracy of plane : 74 %
Accuracy of   car : 72 %
Accuracy of  bird : 23 %
Accuracy of   cat : 44 %
Accuracy of  deer : 55 %
Accuracy of   dog : 34 %
Accuracy of  frog : 56 %
Accuracy of horse : 66 %
Accuracy of  ship : 63 %
Accuracy of truck : 60 %
```

PyTorch

Tensors

- primary data structure used in deep learning
- multidimensional data arrays
- Sample is Scalar

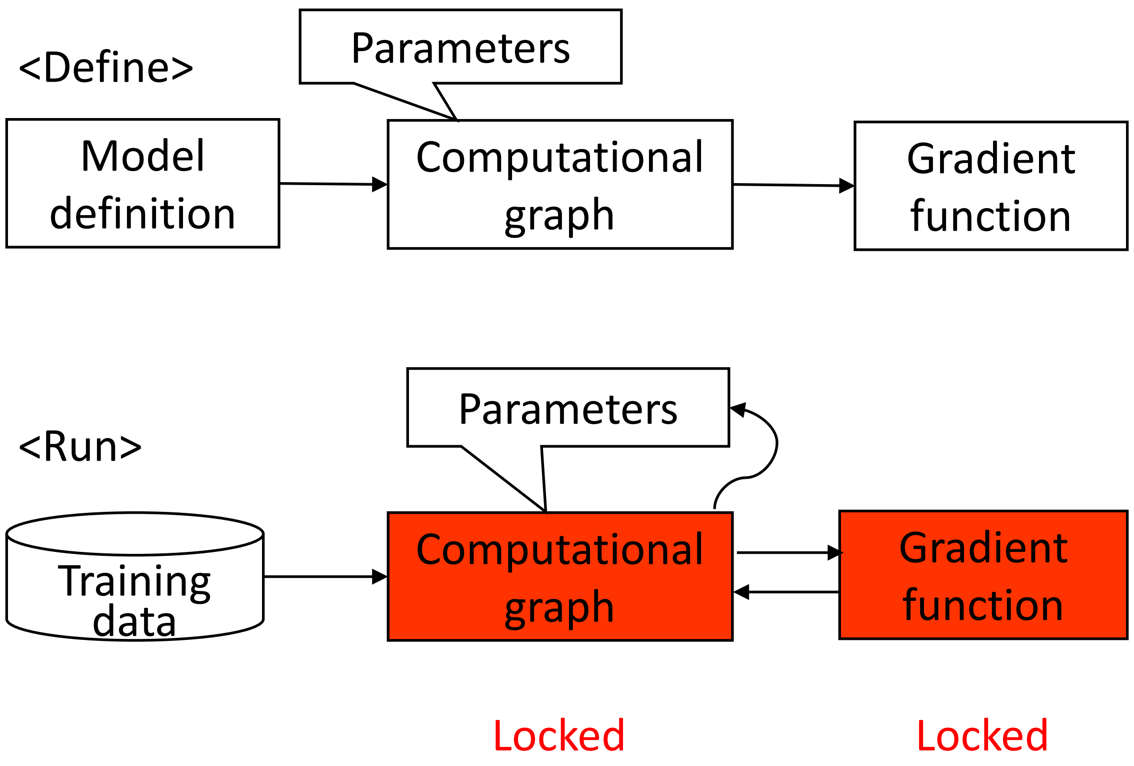
Data	Tensor
Vector	1D tensors of shape (features)
Timeseries	2D tensors of shape (timesteps, features)
Images	3D tensors of shape (height, width, channels)
Video	4D tensors of shape (frames, height, width, channels)



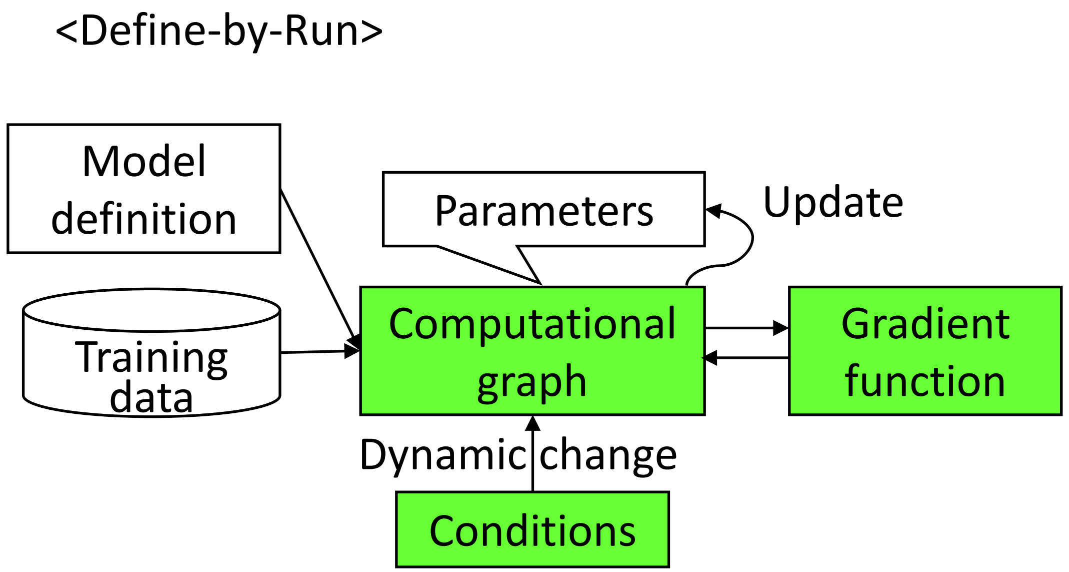
PyTorch

Dynamic Computation Graph

Static Computation Graph: Define-and-Run
(Build graph once, then run many times)



Dynamic Computation Graph: Define-by-Run
(Each forward pass defines a new graph)



Deep Learning Framework

Keras

Standout features

Focus on user experience

Large adoption in the industry

Multi-backend
Multi-platform

Research community

Easy to grasp all concepts

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Import Libraries

Written In R	*Written in Python*
library(keras)	Import numpy as np import pandas as pd import matplotlib.pyplot as plt import keras from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense, Dropout from keras.utils import to_categorical

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Import MINST Dataset

Written In R	*Written in Python*
mnist <- dataset_mnist() x_train <- mnist$train$x y_train <- mnist$train$y x_test <- mnist$test$x y_test <- mnist$test$y	(x_train, y_train), (x_test, y_test) = mnist.load_data()

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Preprocess Train Set and Test Set and Scale the Input Features

Written In R	*Written in Python*
# Reshaping & Scaling the input and output dimensions	# Reshaping & Scaling the input and output dimensions
x_train <- array_reshape(x_train, c(nrow(x_train), 784))/255	x_train=np.reshape(x_train,(x_train.shape[0],-1))/255
x_test <- array_reshape(x_test, c(nrow(x_test), 784))/255	x_test =np.reshape(x_test,(x_test.shape[0],-1))/255
# Performing one-hot encoding on target variables	# Performing one-hot encoding on target variables
y_train <- to_categorical(y_train, 10)	y_train = to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)	y_test = to_categorical(y_test, 10)

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Define Layers to this Model

 # Sequential Model with One Input Layer[784 neurons], 1 Hidden Layer[256 neurons] with Dropout Rate 0.4, 1 Hidden Layer[128 neurons] with Dropout Rate 0.3, and 1 Output Layer [10 #neurons]

Written In R

```r
# Defining the model and layers
model <- keras_model_sequential()
model %>%
   layer_dense(units = 256, activation = 'relu’,
                                              input_shape = c(784)) %>%
   layer_dropout(rate = 0.4) %>%
   layer_dense(units = 128, activation = 'relu') %>%
   layer_dropout(rate = 0.3) %>%
   layer_dense(units = 10, activation = 'softmax')
```

Written in Python

```python
# Defining the model and layers
model = Sequential()
model.add(Dense(256, activation='relu', input_shape=(784,) ))
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(10, activation='softmax'))
```

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Display All Model Layers

Written In R	*Written in Python*
summary(model)	model.summary()

Written In R:

```
summary(model)

________________________________________________________________________________
Layer (type)                      Output Shape                      Param #
================================================================================
dense (Dense)                     (None, 256)                       200960
________________________________________________________________________________
dropout (Dropout)                 (None, 256)                       0
________________________________________________________________________________
dense_1 (Dense)                   (None, 128)                       32896
________________________________________________________________________________
dropout_1 (Dropout)               (None, 128)                       0
________________________________________________________________________________
dense_2 (Dense)                   (None, 10)                        1290
================================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
________________________________________________________________________________
```

Written in Python:

```
model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               200960
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
_________________________________________________________________
```

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Specify the loss function and optimizer to use

Using Adam optimizer and accuracy as metric

Written In R	*Written in Python*

`model %>% compile(loss='categorical_crossentropy',`
`optimizer = optimizer_adam(),`
`metrics = c('accuracy'))`
	`model.compile(loss='categorical_crossentropy',`
	`optimizer="adam",`
	`metrics=['accuracy'])`

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Train the Model and Perform Validation

Written In R	*Written in Python*

`history <- model %>% fit(x_train, y_train,`
`epochs = 30, batch_size = 512,`
`validation_split = 0.2)`
	`history = model.fit(x_train, y_train,`
	`epochs=30,batch_size=512,`
	`validation_split=0.2)`

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Training Sequential Model Result
(Output by R)

```
Epoch 21/30
48000/48000 [==============================] - 3s 71us/sample - loss: 0.0468 - acc: 0.9848 - val_loss: 0.0763 -
val_acc: 0.9797
Epoch 22/30
48000/48000 [==============================] - 3s 63us/sample - loss: 0.0464 - acc: 0.9845 - val_loss: 0.0778 -
val_acc: 0.9788
Epoch 23/30
48000/48000 [==============================] - 3s 64us/sample - loss: 0.0440 - acc: 0.9852 - val_loss: 0.0759 -
val_acc: 0.9788
Epoch 24/30
48000/48000 [==============================] - 3s 61us/sample - loss: 0.0428 - acc: 0.9861 - val_loss: 0.0795 -
val_acc: 0.9781
Epoch 25/30
48000/48000 [==============================] - 3s 66us/sample - loss: 0.0402 - acc: 0.9865 - val_loss: 0.0804 -
val_acc: 0.9785
Epoch 26/30
48000/48000 [==============================] - 3s 63us/sample - loss: 0.0392 - acc: 0.9871 - val_loss: 0.0744 -
val_acc: 0.9804
Epoch 27/30
48000/48000 [==============================] - 3s 65us/sample - loss: 0.0371 - acc: 0.9875 - val_loss: 0.0764 -
val_acc: 0.9799
Epoch 28/30
48000/48000 [==============================] - 3s 60us/sample - loss: 0.0357 - acc: 0.9882 - val_loss: 0.0783 -
val_acc: 0.9791
Epoch 29/30
48000/48000 [==============================] - 3s 56us/sample - loss: 0.0343 - acc: 0.9885 - val_loss: 0.0800 -
val_acc: 0.9794
Epoch 30/30
25600/48000 [===============>..............] - ETA: 1s - loss: 0.0338 - acc: 0.9887[I 15:06:21.615 NotebookApp]
Saving file at /state-of-the-art/rkeras_minst.ipynb
48000/48000 [==============================] - 3s 59us/sample - loss: 0.0338 - acc: 0.9886 - val_loss: 0.0778 -
val_acc: 0.9796
[I 15:08:26.673 NotebookApp] Saving file at /state-of-the-art/rkeras_minst.ipynb
```

Training Sequential Model Result
(Output by Python)

```
Epoch 21/30
48000/48000 [==============================] - 3s 56us/step - loss: 0.0461 - acc: 0.9859 - val_loss: 0.0752 - val_a
cc: 0.9795
Epoch 22/30
48000/48000 [==============================] - 3s 58us/step - loss: 0.0469 - acc: 0.9854 - val_loss: 0.0781 - val_a
cc: 0.9788
Epoch 23/30
48000/48000 [==============================] - 3s 61us/step - loss: 0.0439 - acc: 0.9857 - val_loss: 0.0792 - val_a
cc: 0.9785
Epoch 24/30
48000/48000 [==============================] - 3s 59us/step - loss: 0.0428 - acc: 0.9865 - val_loss: 0.0746 - val_a
cc: 0.9787
Epoch 25/30
48000/48000 [==============================] - 3s 64us/step - loss: 0.0400 - acc: 0.9869 - val_loss: 0.0786 - val_a
cc: 0.9793
Epoch 26/30
48000/48000 [==============================] - 3s 60us/step - loss: 0.0384 - acc: 0.9867 - val_loss: 0.0789 - val_a
cc: 0.9799
Epoch 27/30
48000/48000 [==============================] - 3s 61us/step - loss: 0.0368 - acc: 0.9884 - val_loss: 0.0780 - val_a
cc: 0.9800
Epoch 28/30
48000/48000 [==============================] - 3s 64us/step - loss: 0.0361 - acc: 0.9883 - val_loss: 0.0772 - val_a
cc: 0.9802
Epoch 29/30
48000/48000 [==============================] - 3s 61us/step - loss: 0.0345 - acc: 0.9885 - val_loss: 0.0777 - val_a
cc: 0.9803
Epoch 30/30
48000/48000 [==============================] - 3s 64us/step - loss: 0.0327 - acc: 0.9891 - val_loss: 0.0770 - val_a
cc: 0.9804
```

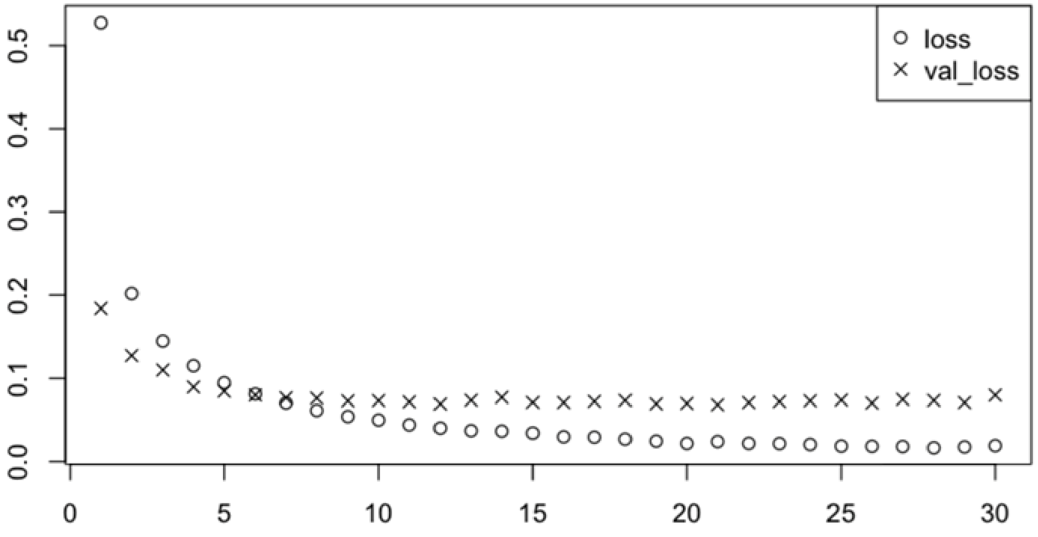
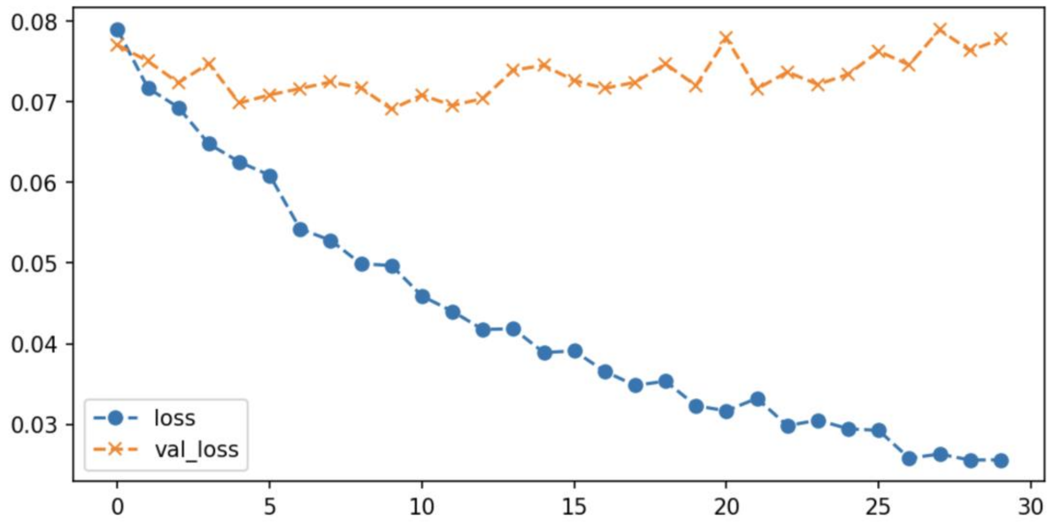
Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Plot the Learning curves for this model

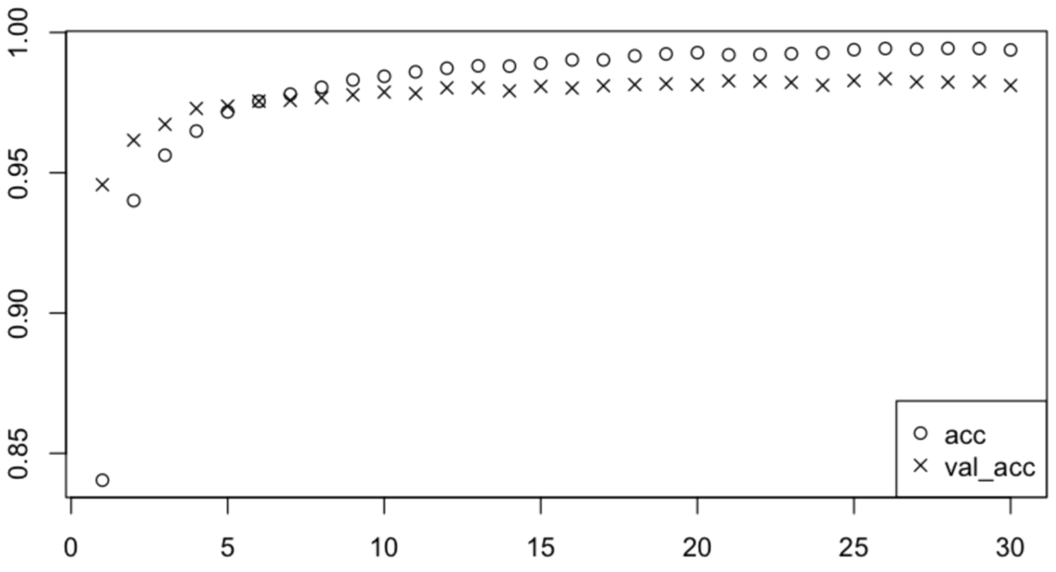
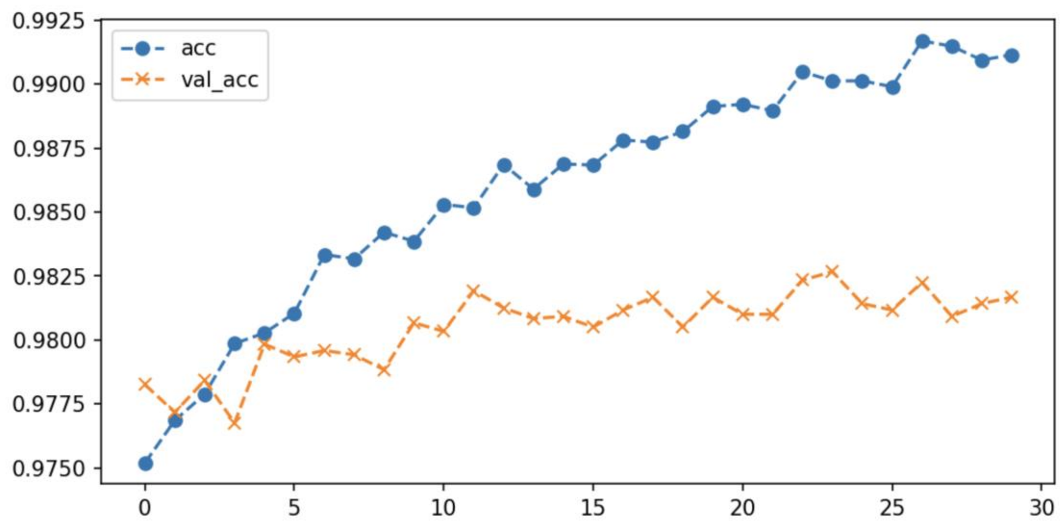
Loss Plot and Accuracy Plot

Written In R	*Written in Python*
plot(history)	df_hist= pd.DataFrame(history.history) plt.plot(df_hist['loss'],'--o') plt.plot(df_hist['val_loss'], '--x') plt.legend(['loss', 'val_loss’]) ; plt.show() plt.plot(df_hist['acc'],'--o') plt.plot(df_hist['val_acc'],'--x') plt.legend(['acc', 'val_acc’]) ; plt.show()

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Learning Curves – Loss Plot (Output by R)	*Learning Curves – Loss Plot (Output by Python)*
	

Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

Learning Curves – Accuracy Plot (Output by R)	*Learning Curves – Accuracy Plot (Output by Python)*
	

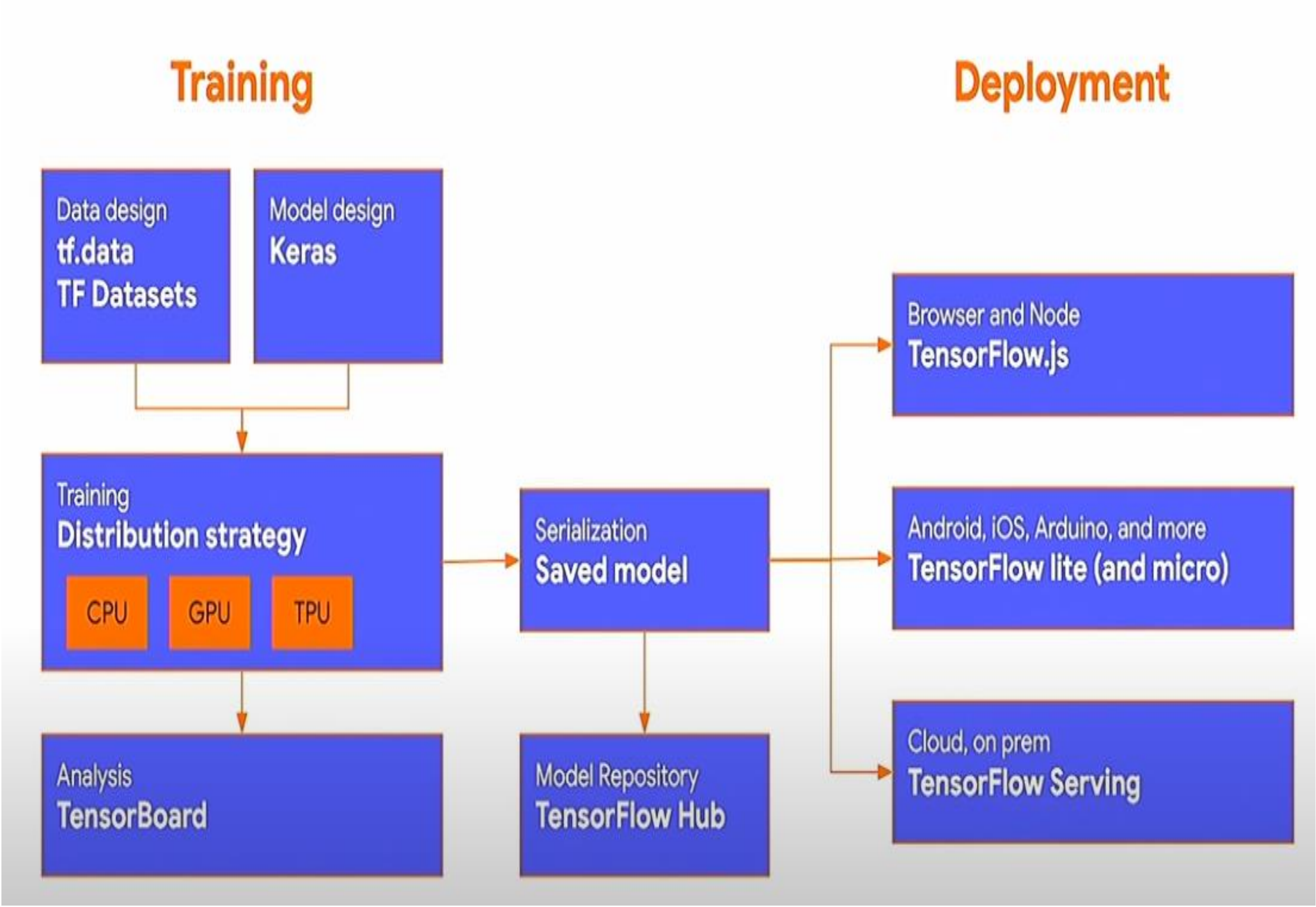
Deep Learning Framework
Code Study 2: Create Keras Model in TensorFlow 2.0

\# **Evaluate the model's performance on the test data**

Written In R	*Written in Python*
loss_acc_metrics <- model %>% evaluate(x_test, y_test) print(loss_acc_metrics) Out: $loss [1] 0.0683236 $acc [1] 0.9832	loss_acc_metrics = model.evaluate(x_test, y_test) print(loss_acc_metrics) Out: [0.07049376319068433, 0.9824]

Deep Learning Framework

TensorFlow 2.0



Major features and improvements

- Keras will be the high-level API for TensorFlow, and it's extended that all the advanced features of TensorFlow can be used directly from tf.keras.
- Eager execution is now the default.
- TensorBoard integration with Keras
- Deep learning researchers will benefit from a low-level API which enables them to export internally used ops and continue to build models onto the internals of TensorFlow without having to rebuild TensorFlow.

Deep Learning Framework

TensorFlow Model Building Style

From simple to arbitrarily flexible

Sequential API
+ built-in layers

New users,
simple models

Models not support:
- share layers
- branches
- multiple inputs
- multiple output

Functional API
+ built-in layers

Engineers with
standard use cases

- More complex models
- Multiple inputs & output
- Easily Define branches
- Easily share layers

Functional API
+ Custom layers
+ Custom metrics
+ Custom losses

Engineers requiring
increasing control

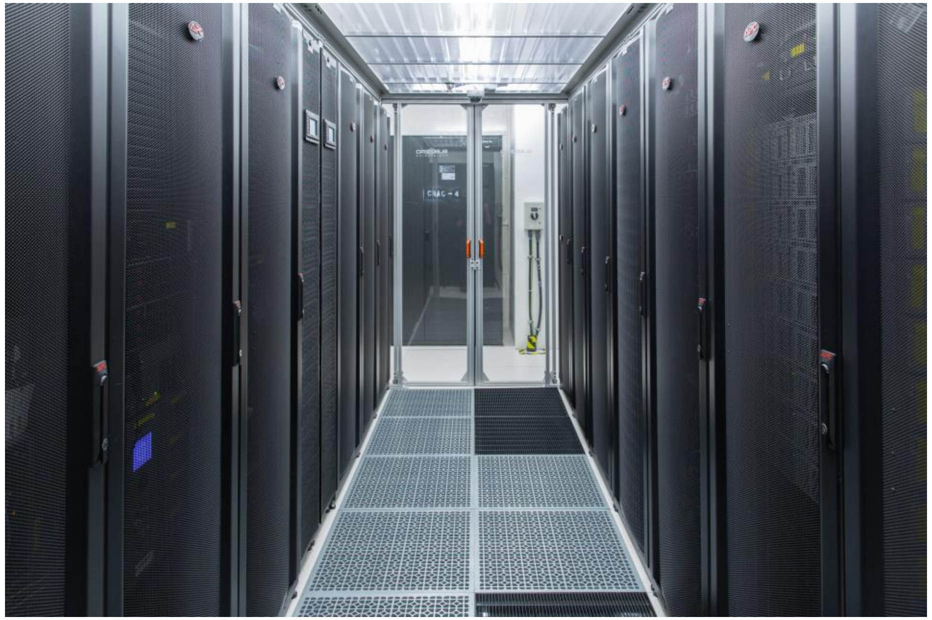
Subclassing:
write everything
from scratch

Researchers

Source: https://goo.gle/TensorFlow

How UBDA support your Deep Learning work?

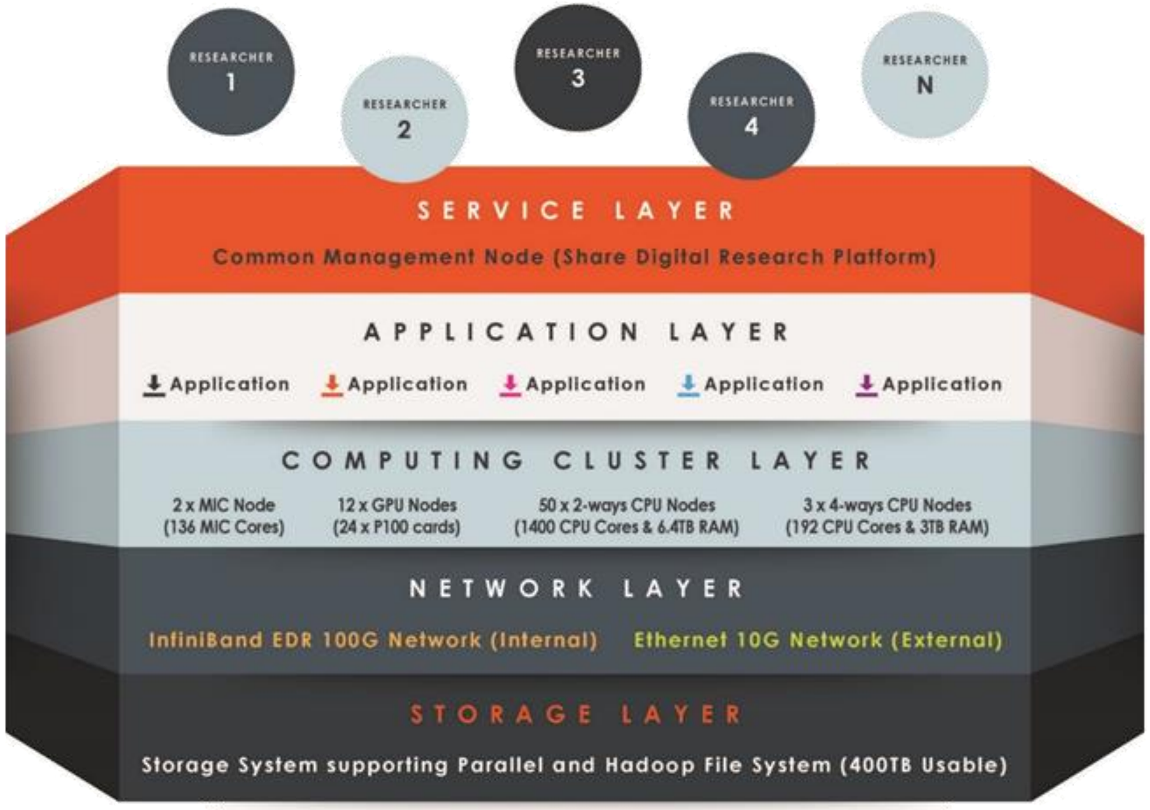
❖ Technical Support



❖ Research Support

UBDA Technical Support
Secure and Scalable Infrastructure

✓ **UBDA provides a dedicated, secure, and scalable 24/7 Linux platform for building deep learning model**



- Storage layer consists of over 400 TB storage system with parallel file systems to allow users to store and process their research data in a reliable environment.
- External network connection is secured with SSH and PolyU Campus Network
- Support users execute their own developed application (C++, Python, R) with multiple (up to 104) CPU cores among multiple computing nodes via MPI
- Support Deep Learning with multiple (up to 4) nVidia P100 GPU Cards
- Allow users install/update libraries as well as software packages in their personal local accounts

UBDA Technical Support
Deep Learning related tool available in UBDA Platform

- Python (supported to *3.8.1*)
- Anaconda (supported to *2019.10*)
- CUDA Toolkit (supported to *10.0*)
- Keras (supported to *2.3.1*)
- PyTorch (supported to *1.4*)
- TensorFlow (support to tensorflow *2.0,* tensorflow-gpu *1.14.0*)
- Intel MPI (support to *2018.0.3*)
- OpenMPI (support to *4.0.1*)
- MPICH (Support to *3.2.1*)

UBDA Technical Support

Numerous job queues for different computing application

- Parallel Programming queue for huge dataset
- Large memory queue for Engineering and Mathematical Modelling
- GPU queue for Deep Learning
- MIC queue for High-Performance Computing (HPC)

UBDA Technical Support
List of Documents support users in platform application

- GUI Files Transfer (Window/Mac iOS)
- UBDA Platform Login (Window/ Mas iOS) and Interface
- UBDA Network Drive Mapping over SSH (Window/Mac iOS)
- **Deep Learning Framework installation with CUDA Support in user environment**
- Job Submission and Scheduling (PBS Scripts)
- Practice with Open-source Python Scripts (e.g. Keras, PyTorch, TensorFlow etc.)
- Job Creation on a Time-based schedule
- Status Information Check
- SSH Keys and Public Key Authentication

UBDA Technical Support
Consultation Service in UBDA Platform Application

Mr. Dillian Wong, Scientific Officer, dillian.wong@polyu.edu.hk

Mr. Jack Wong, Scientific Officer, jack.cw.wong@polyu.edu.hk

Account Registration

Library Installation

Resource Request

Job Submission

UBDA Research Support
Seminar, Talk, Lecture

UBDA Research Support
UBDA Deep Learning Workshop

Topics:

- Deep Learning Algorithms and its Use Cases
- Deep Drive into Recurrent Neural Networks (RNN) model
- Deep Drive into Convolutional Neural Networks (CNN) model
- Deep Learning for Time Series Forecasting
- Unsupervised Deep Learning with Python

UBDA Research Support
UBDA Users Gathering

Aims:

- Experience sharing for empowering users research idea
- enhance the collaborative use and transfer of acquired knowledge of the UBDA platform

UBDA Research Support

Consultation Service in Data-driven Research and Proposal Writing

- Help in analyzing data-driven research problem and formulating the data-driven problem-solving approach
- Support writing proposal related to data-driven research
- Help calculation on the budget involved with using UBDA facility
- Be the supporting party in fund application if necessary

UBDA Research Support

Consultation Service in Data-driven Research and Proposal Writing

Machine Learning

Dr. **Divya SAXENA**, divya.saxena@polyu.edu.hk

Deep Learning

Statistical Analytics

Dr. **Xiao WANG**, xiao-ubda.wang@polyu.edu.hk

Mathematical Optimization

AI

Dr. **Jackei WONG**, jackei.wong@polyu.edu.hk

Big Data

Social Media Analysis

Dr. **Vincent NG**, vincent.ng.comp@polyu.edu.hk

Data Mining

Health Informatics

Thank you !

Question and Answer Section