Navigating the Deep Learning Framework Landscape

A Comprehensive Exploration

Dr. Muhammad Sajjad

RA: Kaleem Ullah

CONTENTS

Pre-Framework Era

Manual Implementation

Developers crafted algorithms from scratch using languages like C, C++, or MATLAB.

Custom Libraries

Some created bespoke tools for specific tasks, lacking broad functionality.

Limited Reusability

Code reuse was minimal, leading to duplicated effort.

High Barrier to Entry

Expertise in algorithms and programming languages was necessary.

Lack of Standardization

Absence of common tools hindered collaboration and progress.

Framework

Definition

A structured set of concepts, practices, and tools for developing algorithms and applications in computer vision.

Components

Includes pre-written code libraries, reusable components, and specialized APIs for image processing tasks.

Purpose

Streamlines development by abstracting low-level image processing tasks, enabling faster prototyping.

General vs Specialized

Can be general-purpose or tailored for specific tasks like object detection or medical imaging.

Benefits

Encourages clean, scalable code through standardized tools and best practices.

Examples

OpenCV: Comprehensive library for image processing and computer vision. –

PyTorch: Flexible deep learning framework for implementing complex neural networks. –

framework

Advantages of Deep Learning Frameworks

Efficiency

Optimized implementations for faster computations.

Community Support

Access to a large community for collaboration.

Abstraction

Simplifies model design and experimentation.

Flexibility

Allows for easy prototyping and customization.

Scalability

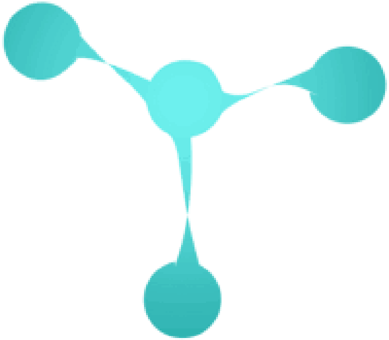
Capable of handling large datasets and models.

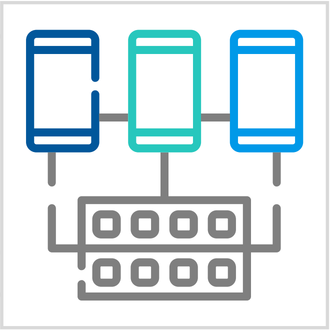
State-of-the-Art Models

Offers pre-trained models for various tasks.

Torch: Forging the Path in Deep Learning Advancement

2002





Purpose

Enabling efficient and scalable symbolic mathematical computation.

Developed by

Pioneered by Ronan Collobert and his team at Facebook AI Research.

Target

Designed to empower deep learning research and development.

Key Feature

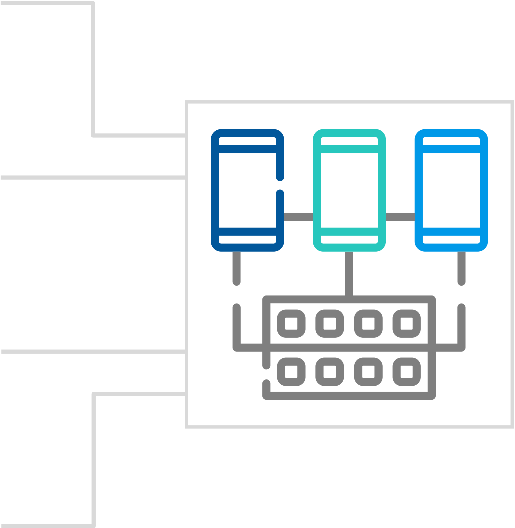
Distinguished by its flexible and modular design for neural network construction.

Performance

Renowned for its optimized implementation and efficient execution of computational graphs.

Theano: Revolutionizing Deep Learning Research

2007



Purpose

Facilitating efficient and scalable symbolic mathematical computation.

Developed by

Led by Yoshua Bengio and a pioneering team at Université de Montréal.

Target

Dedicated to enabling and enhancing deep learning research endeavors.

Key Feature

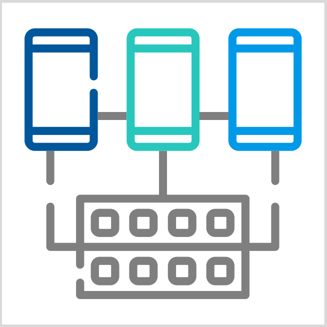
Offering advanced automatic differentiation for seamless neural network training.

Performance

Achieving unparalleled speed and efficiency through meticulous code compilation.

OpenNN: Elevating Deep Learning Exploration

2009



Purpose

Enabling efficient and scalable symbolic mathematical computation.

Developed by

Spearheaded by a visionary team led by [developer's name].

Target

Tailored to empower and elevate deep learning research initiatives.

Key Feature

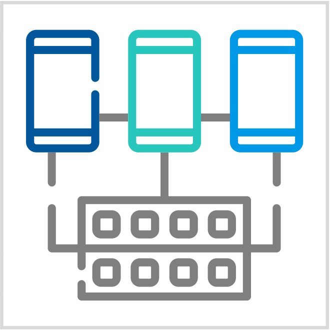
Providing advanced automatic differentiation capabilities for streamlined neural network training.

Performance

Delivering unmatched speed and efficiency through rigorous code optimization.

Caffe: Accelerating Computer Vision Innovation

2013



Purpose

Powering efficient and scalable deep learning model development.

Developed by

Led by Yoshua Bengio and a pioneering team at Université de Montréal.

Target

Tailored for accelerating research and development in computer vision.

Key Feature

Robust support for designing and training convolutional neural networks.

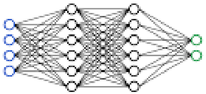
Performance

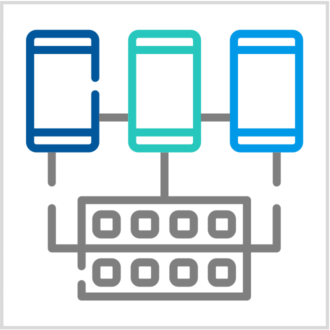
Delivering high-speed processing through optimized GPU utilization.

Keras: Simplifying Deep Learning Development

2015

Keras





Purpose

Facilitating streamlined and scalable development of deep learning models.

Developed by

Spearheaded by François Chollet as part of the TensorFlow project.

Target

Geared towards simplifying the implementation and experimentation of deep learning concepts.

Key Feature

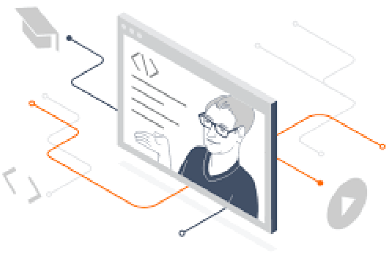
Providing a user-friendly interface and abstraction layer for neural network construction.

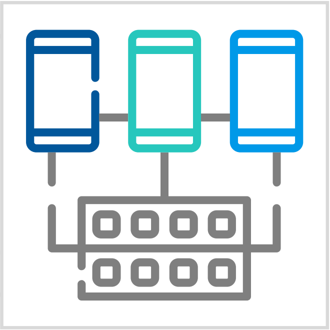
Performance

Ensuring flexibility and performance with seamless integration with TensorFlow backend.

TensorFlow: Empowering Deep Learning Innovation

2015





Purpose

Enabling highly efficient and scalable symbolic mathematical computation.

Developed by

Spearheaded by the Google Brain team led by Jeff Dean and Rajat Monga.

Target

Focused on empowering and advancing deep learning research and applications.

Key Feature

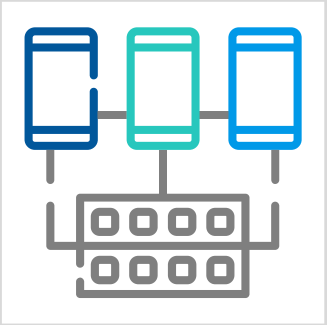
Providing cutting-edge automatic differentiation for seamless and robust neural network training.

Performance

Achieving exceptional speed and efficiency through meticulously optimized code compilation.

MXNet: Empowering Scalable Machine Learning Innovations

2015



Purpose

Enabling efficient and scalable symbolic mathematical computation for machine learning tasks.

Developed by

Collaboratively developed by researchers from multiple institutions including the University of Washington and Carnegie Mellon University.

Target

Geared towards facilitating cutting-edge research and practical applications in deep learning.

Key Feature

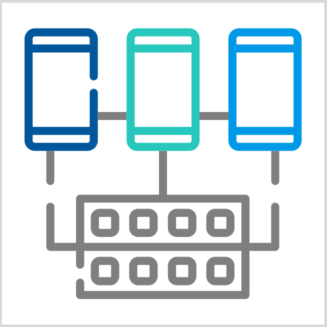
Notable for its flexible programming interface and support for distributed computing, enhancing scalability and performance.

Performance

Demonstrating exceptional speed and efficiency, particularly in distributed computing environments, due to its optimized code compilation.

Chainer: Revolutionizing Deep Learning

2015



Purpose

Empowering efficient and scalable symbolic mathematical computations.

Developed by

Spearheaded by a visionary team, including Seiya Tokui, at Preferred Networks.

Target

Geared towards facilitating and elevating deep learning research and development.

Key Feature

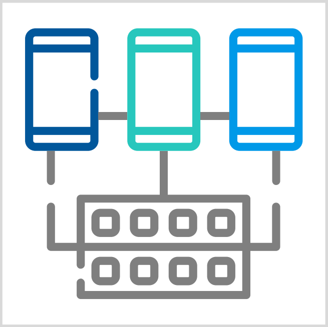
Introducing innovative dynamic computation graph for agile neural network training.

Performance

Delivering exceptional speed and efficacy via rigorous code optimization.

PyTorch: Redefining Deep Learning Innovation

2016



Purpose

Empowering efficient and scalable symbolic computation for neural networks.

Developed by

Spearheaded by a pioneering team led by researchers at Facebook AI.

Target

Aimed at revolutionizing deep learning research and application development.

Key Feature

Introducing dynamic computation graph for flexible model design and debugging.

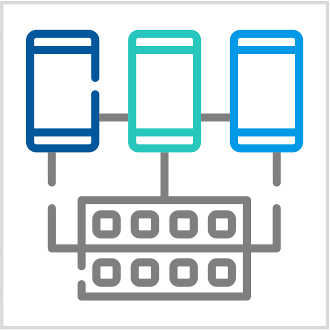
Performance

Delivering superior speed and efficiency through optimized tensor computations.

Microsoft Cognitive Toolkit (CNTK): Empowering AI Exploration

2016

Microsoft Cognitive Toolkit



Purpose

Empowering efficient and scalable symbolic mathematical computation.

Developed by

Spearheaded by a visionary team at Microsoft Research.

Target

Committed to empowering and elevating deep learning research initiatives.

Key Feature

Providing cutting-edge automatic differentiation for seamless neural network training.

Performance

Attaining unmatched speed and efficiency via rigorous code compilation.

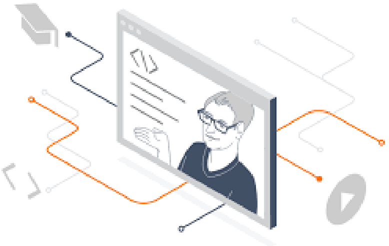
TenserFlow

Origins

- TensorFlow was developed in November 2015 by a team of researchers and engineers at Google's Brain team.
- It was created to meet the increasing need for a flexible and scalable open-source machine learning framework.
- The motivation behind TensorFlow's creation was to provide a platform that could facilitate the development and deployment of machine learning models across various domains and applications.

Development

- The development of TensorFlow was led by the Google Brain team, comprising experts in machine learning, software engineering, and data science.
- Visionaries like Jeff Dean and Rajat Monga played key roles in spearheading the project.
- The team's collaborative efforts resulted in the release of TensorFlow, which quickly gained traction and became one of the most widely used machine learning frameworks globally.



Significance

- TensorFlow revolutionized the field of artificial intelligence by offering extensive support for deep learning and neural networks.
- It empowered researchers and developers with a powerful tool for building and deploying cutting-edge machine learning models efficiently.
- TensorFlow's inception marked a significant milestone in the advancement of machine learning, laying the groundwork for numerous innovations and advancements in AI and related fields.

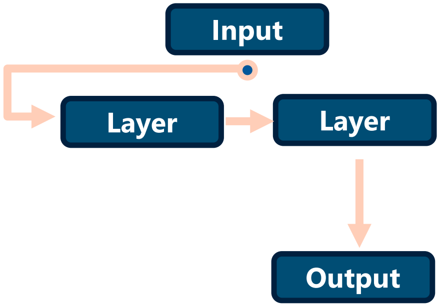
Modeling

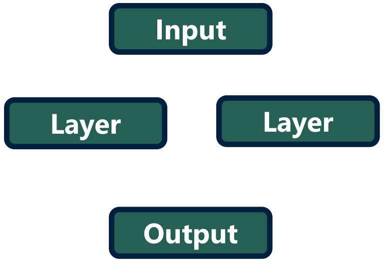
Sequential API

- The Sequential API offers a simple and intuitive approach for building neural networks, particularly suited for beginners and standard architectures.
- It enables you to construct models layer by layer in a linear fashion, with each layer feeding its output to the next layer.
- This method is well-suited for creating straightforward architectures, including feedforward neural networks and convolutional neural networks (CNNs).

Functional API

- The Functional API offers a flexible and powerful approach for building neural networks, ideal for complex architectures and advanced functionalities.
- It enables the creation of models with multiple input and output layers, shared layers, and branched architectures.
- This method is preferred for constructing models with intricate connections and non-linear network structures.





TensorFlow vs Keras

TensorFlow

- TensorFlow is a comprehensive open-source machine learning framework developed by Google.
- It provides a wide range of tools and functionalities for building, training, and deploying machine learning models.
- TensorFlow offers flexibility and scalability, catering to various applications, from research to production deployment.
- It allows for low-level control over model architecture and optimization, ideal for advanced users and complex projects.
- TensorFlow supports not only neural networks but also other machine learning algorithms and techniques.

TensorFlow

Keras

Keras

- Keras is a high-level neural networks API, initially developed independently.
- It offers a user-friendly interface for building and training neural networks with minimal code.
- Emphasizing simplicity and ease of use, Keras is ideal for beginners and rapid prototyping.
- Although Keras can be used independently, it has been integrated into TensorFlow as its official API since TensorFlow version 2.0.
- Within TensorFlow, Keras maintains its user-friendly features while harnessing TensorFlow's scalability and performance.

TensorFlow vs PyTorch

TensorFlow

- TensorFlow, developed by Google, is a robust and flexible open-source machine learning framework.
- It provides extensive support for deep learning and neural networks, offering a diverse array of pre-built models and tools.
- TensorFlow is renowned for its scalability and readiness for production deployment, making it a popular choice in both research and industry.
- Utilizing a static computational graph, TensorFlow defines the graph structure before execution, enabling optimizations like graph compilation and distributed execution.

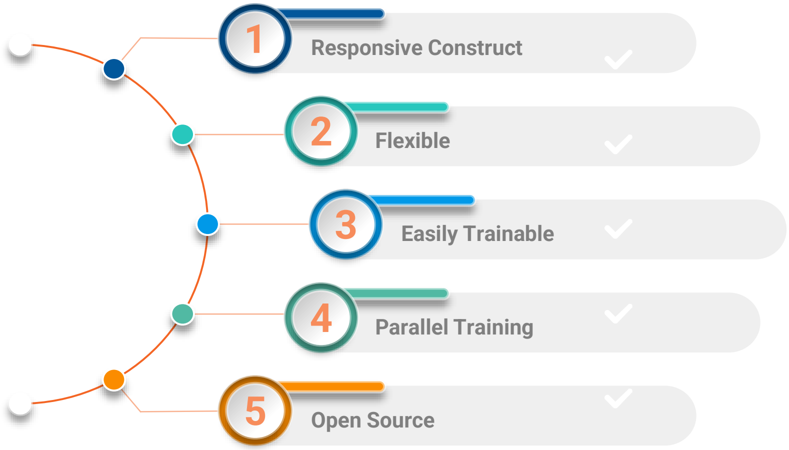
TensorFlow

PyTorch

PyTorch

- Developed by Facebook's AI Research lab, PyTorch is a dynamic deep learning framework celebrated for its simplicity and flexibility.
- PyTorch boasts a dynamic computational graph, facilitating intuitive model building and debugging.
- Favored by researchers and academics for its ease of use and robust support for dynamic computation, PyTorch excels in experimentation and research projects.
- PyTorch's imperative programming style enables easy debugging and experimentation, allowing models to be built and modified on the fly.

Features



Efficient execution across hardware platforms.

Allows custom neural network architectures.

Simplifies model training with high-level APIs.

Supports distributed training for speed.

Accessible source code for customization.

Continue

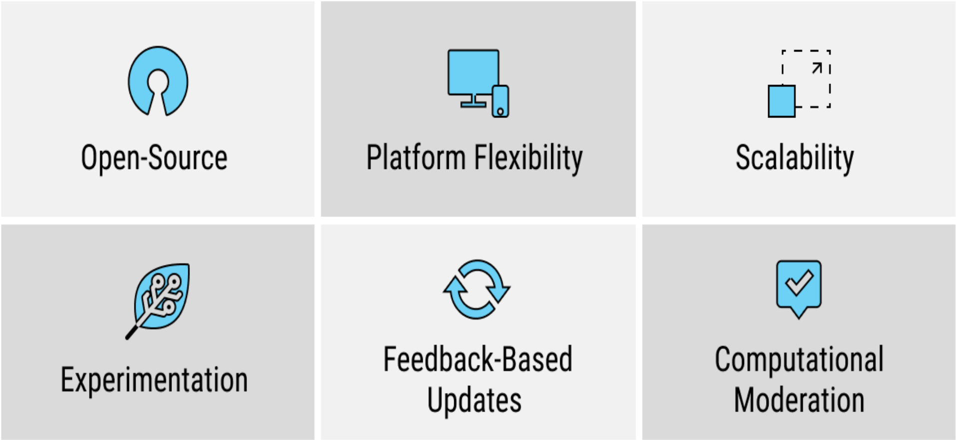
- 6 **Feature Columns** — Tools for preprocessing structured data.
- 7 **Layered Components** — Modular architecture for flexibility.
- 8 **Statistical Distributions** — Includes various distributions for modeling uncertainty.
- 9 **Layered components** — Modular architecture for flexibility.
- **Visualizer (with Tensorboard)** — Visualization tool for monitoring models.

Overcoming Framework Limitations with Keras

- ❑ Keras resolved limitations in flexibility and customization of earlier frameworks.
- ❑ Provided a high-level, user-friendly API for building neural networks.
- ❑ Streamlined development process for rapid prototyping and experimentation.

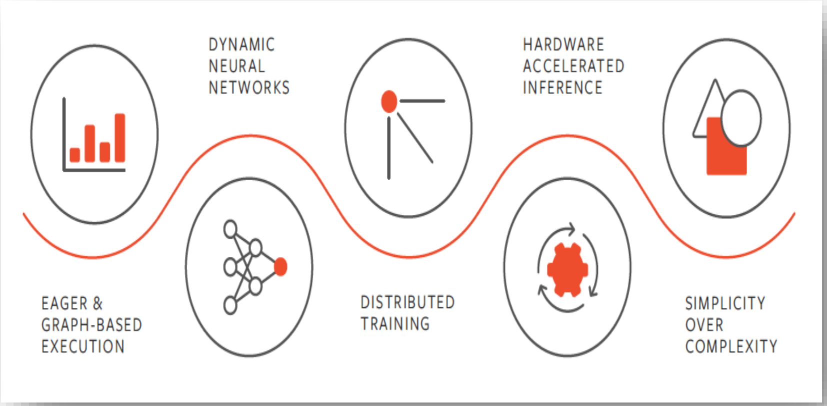
Overcoming Framework Limitations with TensorFlow

- TensorFlow addressed challenges of complex API and steep learning curve.
- Introduction of higher-level APIs like Keras enhanced accessibility.
- Improved productivity in building and deploying deep learning models.

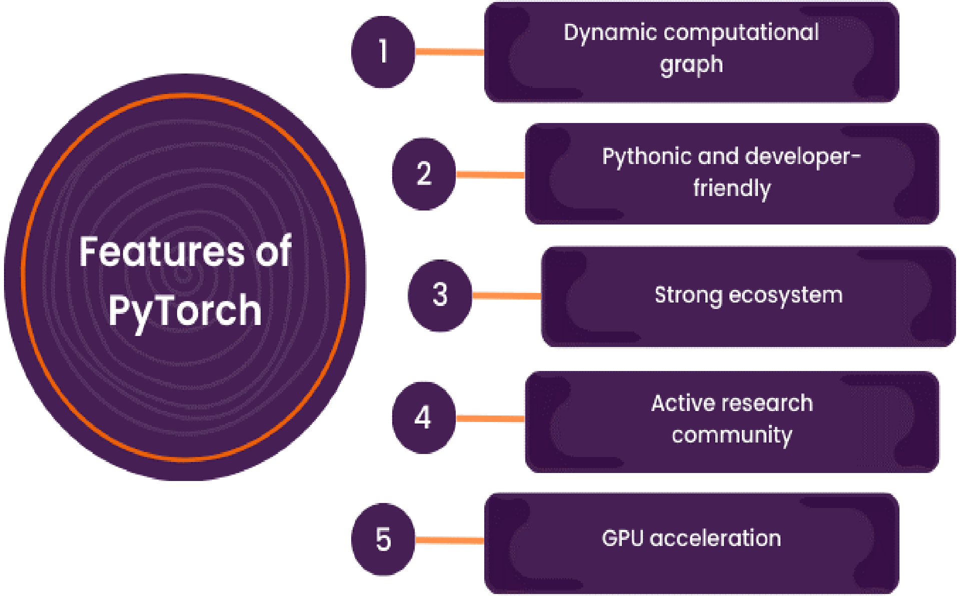


Overcoming Framework Limitations with PyTorch

- ❑ Dynamic computational graphs.
- ❑ Can make use of standard Python flow control.
- ❑ Support for Python debuggers.
- ❑ It is liked in research area.
- ❑ Might not be as efficient as TensorFlow for large training tasks.



PyTorch: The Choice of Researchers



Keras

- Keras is an open-source deep learning framework primarily designed for fast experimentation and prototyping of neural networks.

- It is a high-level neural networks API written in Python, facilitating easy construction, training, and deployment of neural networks, and compatible with TensorFlow, CNTK, or Theano.

- It can be run on both CPU and GPU.

- It was developed by François Chollet and first released in March 2015.

Advantages of Keras

Keras offered unique advantages, setting it apart from other frameworks, such as:

Simplicity and Ease of Use

- ❑ Minimal boilerplate code for quick prototyping.

Extensive Backend Support:

- ❑ Compatible with multiple backend engines such as TensorFlow etc.
- ❑ Offers flexibility to choose backend based on project requirements and preferences.

Modularity and Flexibility

- Modular design facilitates construction of complex architectures.
- Encourages experimentation with different network configurations.

High-Level Abstractions

- Provides intuitive abstractions for common deep learning tasks.
- Simplifies implementation of complex algorithms.

Community and Ecosystem

- Keras has been open-source since its initial release
- Large and active community of users and contributors.
- Rich ecosystem of libraries, tools, and resources for support and extension.

Integration of Keras into TensorFlow

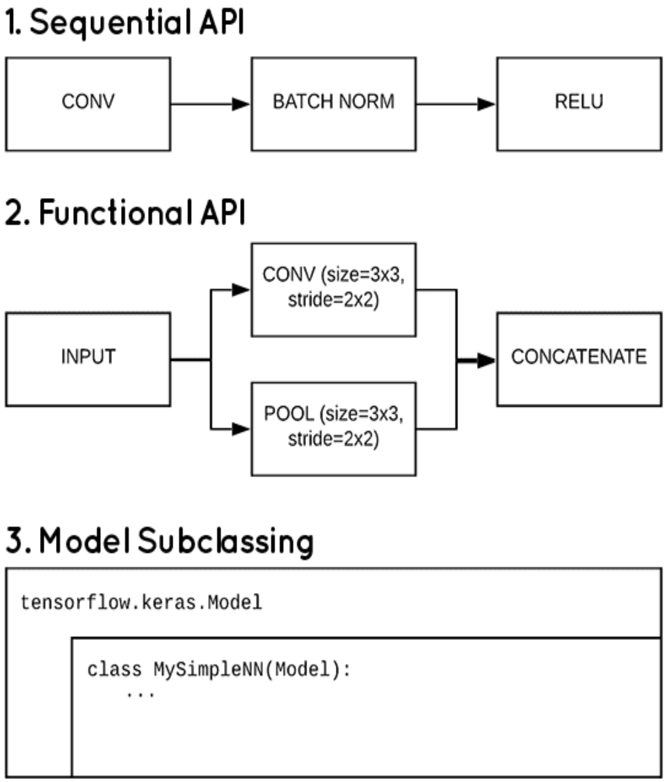
- ❑ Google's deep learning framework TensorFlow integrated Keras into its core library in 2017.
- ❑ Keras was developed and is maintained by Francois Chollet and is part of the Tensorflow core, which makes it Tensorflow’s preferred high-level API.
- ❑ This integration enabled users to utilize Keras as a high-level interface.
- ❑ Users could leverage TensorFlow's powerful features as the backend.
- ❑ Latest versions: TensorFlow: *tensorflow 2.16. 1,* Keras *keras 3.3*
- ❑ Now we can import keras either as standalone or as part of TensorFlow
  - ➢ import keras
  - ➢ from tensorflow import keras

Creating models in Keras:

Sequential Model:

- ❑ Simplest way to create a model in Keras, where layers are added sequentially.
- ❑ Suitable for most simple architectures such as feedforward neural networks and convolutional neural networks (CNN)
- ❑ The problem with the sequential API is that it doesn't allow models to have multiple inputs or outputs, which are needed for some problems.

```
# Define Sequential model with 3 layers
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
```



Functional API:

- It allows for more flexibility in model architectures.
- It enables the creation of complex models with multiple inputs, multiple outputs, shared layers, and branching topologies.

```
# Define input layer
inputs = Input(shape=(input_shape,))

# Define layers
layer1 = Dense(2, activation="relu", name="layer1")(inputs)
layer2 = Dense(3, activation="relu", name="layer2")(layer1)
layer3 = Dense(4, name="layer3")(layer2)

# Define output
outputs = layer3

# Create functional model
model = Model(inputs=inputs, outputs=outputs)
```

Conventions in Keras

The conventions ensure consistency and readability in code, facilitating collaboration and maintenance.

1. Sequential Naming:

Layers named sequentially (e.g., layer1, layer2) for easy identification.

2. Meaningful Layer Names:

Give layers meaningful names using the **name** parameter (e.g., dense_layer1).

3. Consistent Activation Functions:

Use consistent activation functions throughout the model for coherence.

4. Variable Naming:

Use descriptive variable names to indicate their purpose (e.g., input_data, learning_rate)

5. Clear Documentation:

Include comments and docstrings for clear documentation of functions and code blocks.

Comparisons:

Keras vs TensorFlow

Keras:

- Utilizes static computational graphs.
- Focuses on simplicity and ease of use.
- High-level abstraction for rapid prototyping.
- Ideal for quick experimentation and model iteration.

TensorFlow:

- Offers both static and dynamic computational graphs.
- Provides extensive control and flexibility.
- Widely adopted for production-level deployment.
- Known for scalability and performance in complex tasks.

Keras vs PyTorch:

Keras

- Utilizes static computational graphs.
- Limited use of standard Python flow control.
- Limited support for Python debuggers.
- Popular in industry and academia.
- Known for efficiency in large-scale training tasks.

PyTorch

- Dynamic computational graphs.
- Can make use of standard Python flow control.
- Support for Python debuggers.
- It is liked in research area.
- Might not be as efficient as TensorFlow for large training tasks.

Visualizing Models in Keras

Model Summary: *summary()*

- Provides a concise overview of model architecture.
- Displays layer types, shapes, and total parameters.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 2)                 4

 dense_10 (Dense)            (None, 1)                 3

 dense_11 (Dense)            (None, 2)                 4

=================================================================
Total params: 11
Trainable params: 11
Non-trainable params: 0
_________________________________________________________________
None
```

Plotting Model Architectures:

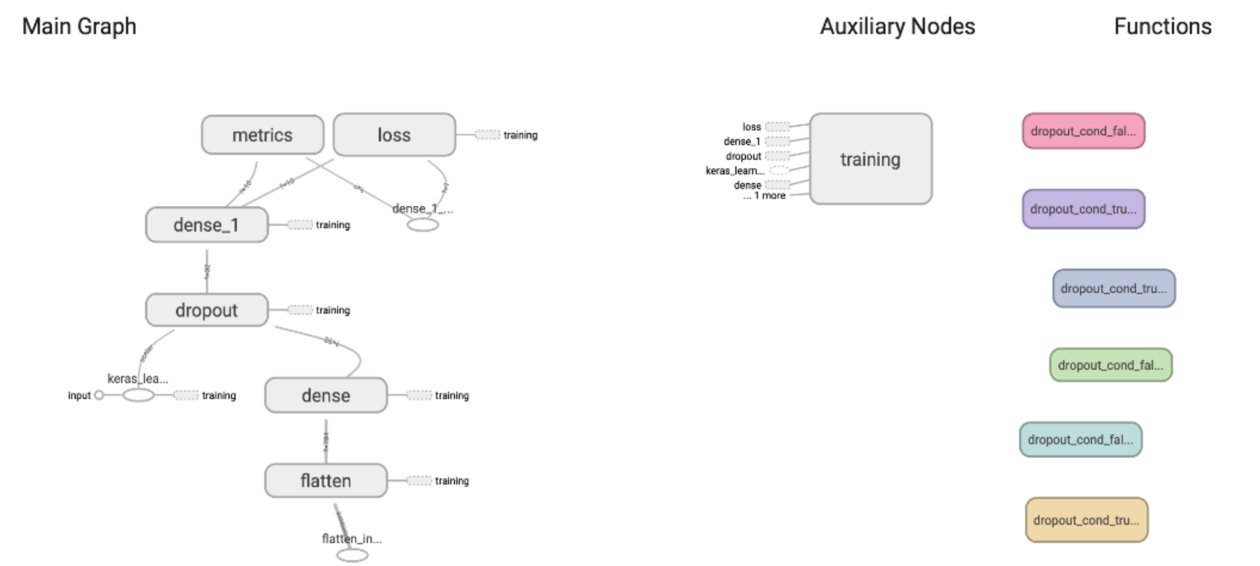
- Utilizes tools like **plot_model**.
- Generates graphical representations of neural network flow.

Visualizing Training History:

- Utilizes Matplotlib to track metrics over epochs.
- Monitors training progress and model performance.

TensorBoard Integration:

- Smoothly integrates with TensorBoard.
- Captures different metrics and model designs during training.



What is PyTorch?

PyTorch is a deep learning framework for building neural networks used in various domains such as computer vision and natural language processing.

- Developed by Facebook's AI Research lab (FAIR)
- Released in October 2016 as an open-source project
- Popular among researchers and practitioners in the deep learning community.
- Used by large companies like Meta, Tesla, Uber, and Nvidia.

Origin of PyTorch

PyTorch originated from **Torch**, a scientific computing framework and machine learning library written in Lua programming language.

What Torch Provided?

- ❑ Provided efficient numerical computations and tensor operations.
- ❑ Provided dynamic computation graphs, enabling researchers to experiment quickly.
- ❑ Built on a C/C++ core, offering speed and efficiency.

What Torch Lacked?

- ❑ Strong integration with Python, which was becoming the de-facto language for machine learning.
- ❑ Torch had a steeper learning curve due to its Lua-based syntax.
- ❑ Its API was not user-friendly.

Origin of PyTorch

The existing frameworks which were popular for deep learning, like **TensorFlow**, have limitations.

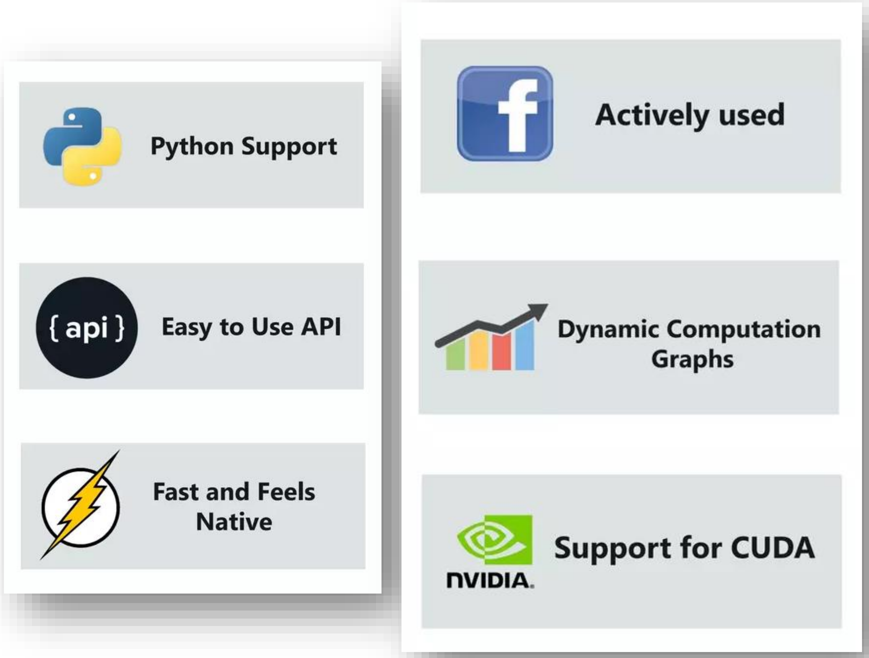
- ❑ Static computation graph doesn’t allow for explicit data movement.
- ❑ Thus less control over low level operations.
- ❑ The API was not user friendly (or Pythonic).

Origin of PyTorch

To address these limitations, **PyTorch** was created.

PyTorch Provided:

- ❑ a Python API to leverage **Torch** capabilities.
- ❑ Tensor operations that are heavily inspired by NumPy.
- ❑ Autograd engine for efficient computation of gradients in neural networks.
- ❑ Integration with CUDA, enabling GPU acceleration

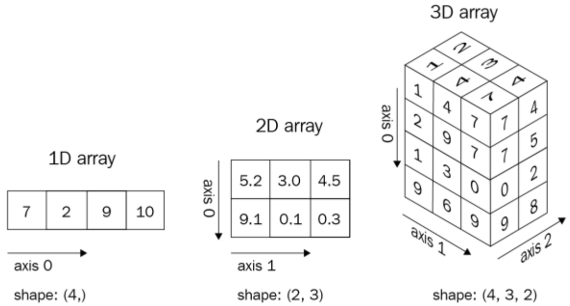


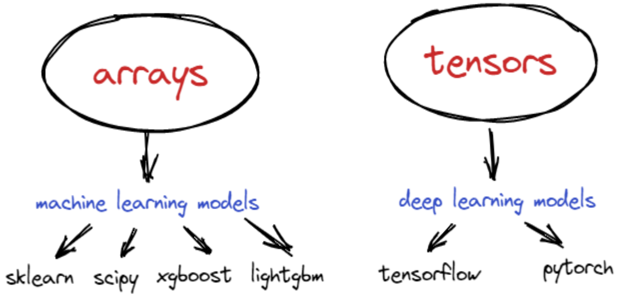
PyTorch Tensors

PyTorch **tensors** are the fundamental building blocks for deep learning tasks. The are similar to NumPy arrays with additional features.

PyTorch Tensors

- ❑ Can reside on GPUs for faster computations.
- ❑ Support automatic differentiation to compute gradients during training.
- ❑ Operations that are supported are very similar to NumPy arrays but are faster.



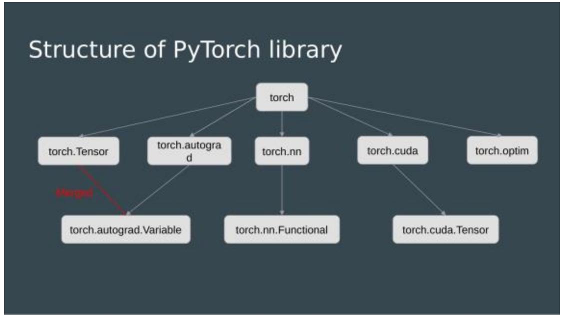


Structure of PyTorch Library

PyTorch provides several modules for deep learning tasks, from loading data to defining layers and choosing optimizers.

PyTorch Modules

- ❑ **torch:** the main module that provides tensors and operations on them.
- ❑ **torch.nn:** includes classes and functions to define layers, loss functions and activation functions.
- ❑ **torch.optim:** provides optimization algorithms for training (SGD, ADAM, etc)
- ❑ **torch.utils.data:** provides functionalities for loading datasets and creating batches during training




```python
import torch
from torch.nn import ReLU, Sigmoid, MSELoss, CrossEntropyLoss
from torch.optim import SGD, Adam
from torch.utils.data import Dataset, DataLoader

# tensor creation
tensor_a = torch.tensor([
    [1, 2],
    [3, 4]
], dtype=torch.int32)

# numpy like operations
tensor_ones = torch.ones(3, 3)
tensor_normal = torch.randn(3, 3)
ones_plus_normal = tensor_ones + tensor_ones
```

Useful PyTorch Packages

PyTorch provides several other package that are specifically designed for tasks such as computer vision, NLP, audio processing, etc.

“torchvision” provides

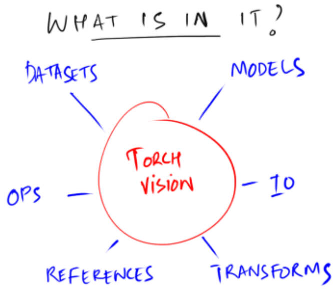
- ❑ Access to popular computer vision datasets, pretrained models, and preprocessing.

“Torchaudio” provides

- ❑ Utilities for loading and preprocessing audio data.
- ❑ Datasets and pretrained models for speech recognition and sound classification.

“torchtext” provides

- ❑ Utilities for loading and preprocessing text data for NLP tasks.
- ❑ Datasets, tokenizers and pretrained word embeddings.



TorchAudio

TorchText

Visualization and Experiment Tracking

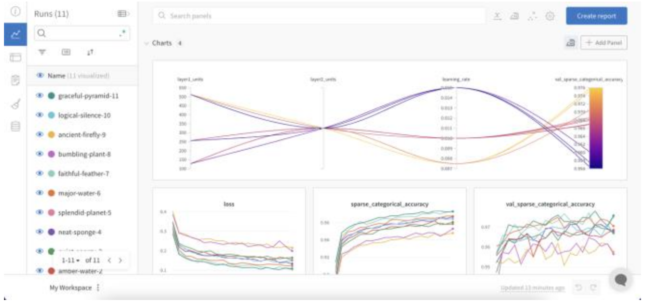
PyTorch integrates well with several popular logging tools for logging the training process and make visualization, such as:

TensorBoard

TensorFlow *TensorBoard* is compatible with PyTorch via the **`tensorboardX`** that is used to visualize training metrics like loss, accuracy, f1 score, etc.

Weights and Biases, Neptune.ai

It is a cloud platform that provides tools for experiment tracking, versioning, visualization, and collaboration.



PyTorch Popularity

PyTorch is increasingly becoming popular in research area for the following reasons.

HuggingFace

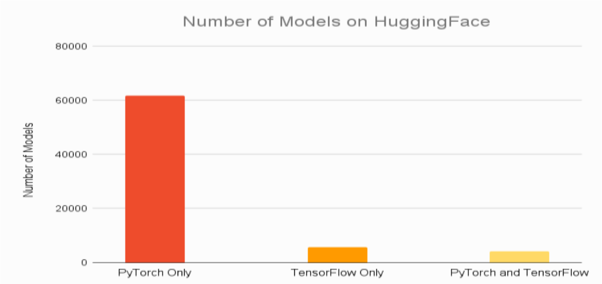
- ❑ 92% of models on HuggingFace are PyTorch exclusive

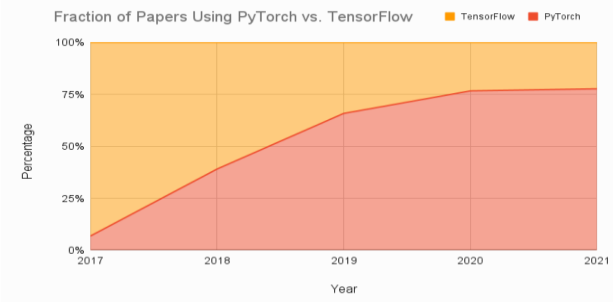
Research Papers

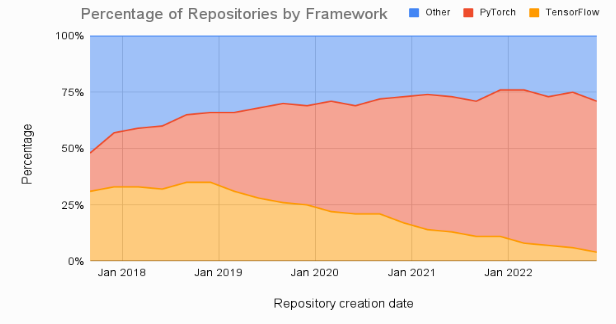
- ❑ Most of the recently published research papers use PyTorch

Papers with Code

- ❑ It is a website that provides machine learning papers with code, almost 70% of them are implemented in PyTorch.







PyTorch vs TensorFlow

PyTorch

- ❑ Provides a Pythonic syntax and interface.
- ❑ Offers flexibility for researchers to control the data movement.
- ❑ Can make use of standard Python flow control.
- ❑ Is popular in research area.
- ❑ Need third party for visualization

TensorFlow

- ❑ Provides static computation graphs.
- ❑ Offers high level interface for to build models.
- ❑ Hard to make quick changes to the model.
- ❑ Cannot make use of standard Python flow control.
- ❑ More mature and preferred for production environment.

PyTorch vs TensorFlow

PyTorch

```python
import torch
import torch.nn as nn
import torch.optim as optim

# Define a simple neural network
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(10, 5)
        self.fc2 = nn.Linear(5, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        return x

# Create an instance of the model
model = SimpleNN()

# Define dummy input and target
input_data = torch.randn(1, 10)
target = torch.randn(1, 1)

# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Training loop
for epoch in range(100):
    optimizer.zero_grad()
    output = model(input_data)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()

print("Training finished!")
```

TensorFlow

```python
import tensorflow as tf

# Define a simple neural network
class SimpleNN(tf.keras.Model):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = tf.keras.layers.Dense(5, activation='relu')
        self.fc2 = tf.keras.layers.Dense(1, activation='sigmoid')

    def call(self, inputs):
        x = self.fc1(inputs)
        x = self.fc2(x)
        return x

# Create an instance of the model
model = SimpleNN()

# Define dummy input and target
input_data = tf.random.normal((1, 10))
target = tf.random.normal((1, 1))

# Define loss function and optimizer
loss_fn = tf.keras.losses.MeanSquaredError()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

# Training loop
for epoch in range(100):
    with tf.GradientTape() as tape:
        output = model(input_data)
        loss = loss_fn(target, output)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

print("Training finished!")
```

Community:

A strong community encourages knowledge sharing, support, collaboration, and continuous improvement, driving the success and advancement of a technology.

- François Chollet, the creator of Keras, actively engages with the community.

- TensorFlow has one of the largest and most active communities in the deep learning ecosystem.

- PyTorch is favored by many researchers for its flexibility, dynamic computation graphs, and intuitive design.

THANK YOU