DY PATIL COLLEGE OF ENGINEERING

FIRST YEAR ENGINEERING DEPARTMENT

THE ACADEMIC YEAR 2022-23 SEMESTER 1

Presenters

43	Tanmay Borase	46	Aryan Kshatriya
44	Yash Kawdiya	47	Rajveersingh Naruka
45	Harshal Kolhe	48	Rugved Salunke

Subject Teacher : Swati Suryawanshi Mam

PYTORCH INTRODUCTION

Group 8 | Division H

CONTENT OVERVIEW

CONTENT OVERVIEW

WHAT IS PYTORCH

PyTorch is an open-source deep learning framework that's known for its flexibility and ease-of-use. This is enabled in part by its compatibility with the popular Python high-level programming language favored by machine learning developers and data scientists.

WHY PYTORCH

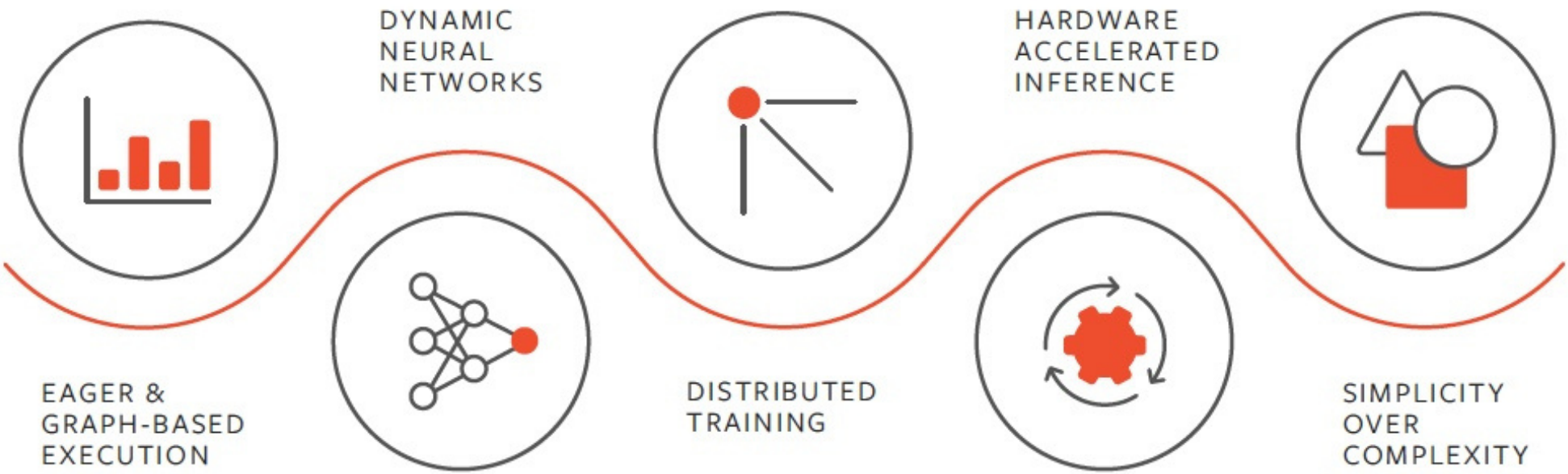


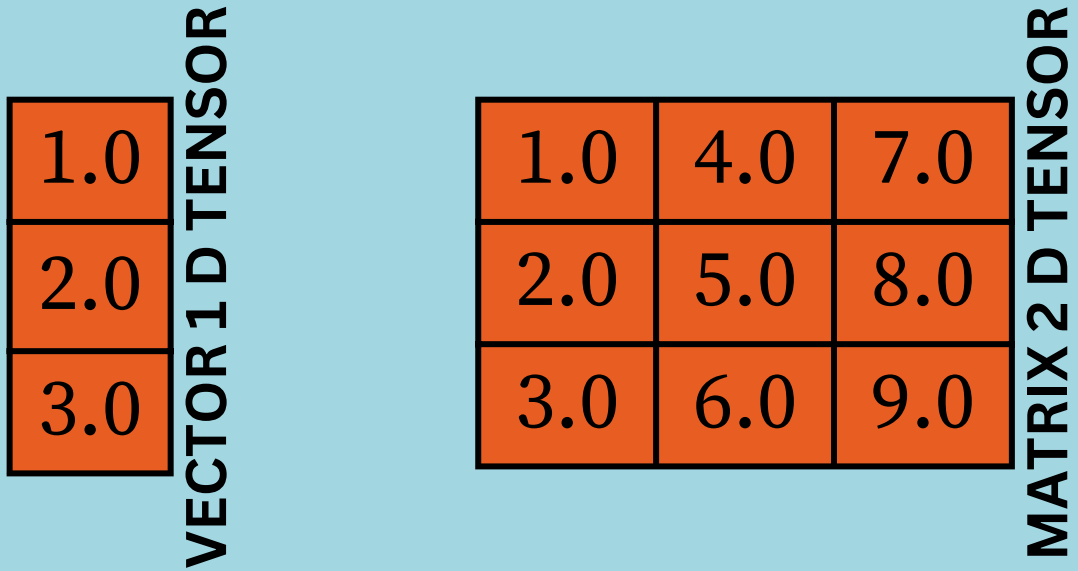
Image reference https://pytorch.org/features/

PyTorch is a fully featured framework for building deep learning models, which is a type of machine learning that's commonly used in applications like image recognition and language processing.

THE PYTORCH EXPLAINED

PYTORCH TENSORS

The Pytorch is used to process the tensors. Tensors are multidimensional arrays like n-dimensional NumPy array. However, tensors can be used in GPUs as well, which is not in the case of NumPy array. PyTorch accelerates the scientific computation of tensors as it has various inbuilt functions.
A vector is a one-dimensional tensor, and a matrix is a two-dimensional tensor. One significant difference between the Tensor and multidimensional array used in C, C++, and Java is tensors should have the same size of columns in all dimensions. Also, the tensors can contain only numeric data types.



The two fundamental attributes of a tensor are:

- **Shape** refers to the dimensionality of array or matrix
- **Rank** refers to the number of dimensions present in tensor

Code	Output

# importing torch 	
import torch
 	
# creating a tensors 	
t1=torch.tensor([1, 2, 3, 4])
t2=torch.tensor([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
 	
# printing the tensors: 	
print("Tensor t1: \n", t1)
print("\nTensor t2: \n", t2)
 	
# rank of tensors 	
print("\nRank of t1: ", len(t1.shape))
print("Rank of t2: ", len(t2.shape))
 	
# shape of tensors 	
print("\nRank of t1: ", t1.shape)
print("Rank of t2: ", t2.shape)
	Tensor t1:
	tensor ([1, 2, 3, 4])
	Tensor t2:
	tensor([[1, 2, 3, 4], [5,6,7,8], [9,10,11,12]])
	Rank of t1: 1
	Rank of t2: 2
	Rank of t1: torch.Size([4])
	Rank of t2: torch.Size([3,4])

CREATING TENSOR IN PYTORCH

There are various methods to create a tensor in PyTorch. A tensor can contain elements of a single data type. We can create a tensor using a python list or NumPy array. The torch has 10 variants of tensors for both GPU and CPU. Below are different ways of defining a tensor.

torch.Tensor()
It copies the data and creates its tensor. It is an alias for torch.FloatTensor.

torch.tensor()
It also copies the data to create a tensor; however, it infers the data type automatically.

torch.as_tensor()
The data is shared and not copied in this case while creating the data and accepts any type of array for tensor creation.

torch.from_numpy()
It is similar to tensor.as_tensor() however it accepts only numpy array.

Code	Output

# importing torch module 	
import torch
import numpy as np
 	
# list of values to be stored as tensor 	
data1 = [1, 2, 3, 4, 5, 6]
data2 = np.array([1.5, 3.4, 6.8, 9.3, 7.0, 2.8])
 	
# creating tensors and printing 	
t1 = torch.tensor(data1)
t2 = torch.Tensor(data1)
t3 = torch.as_tensor(data2)
t4 = torch.from_numpy(data2)
 	
print("Tensor: ",t1, "Data type: ", t1.dtype,"\n")
print("Tensor: ",t2, "Data type: ", t2.dtype,"\n")
print("Tensor: ",t3, "Data type: ", t3.dtype,"\n")
print("Tensor: ",t4, "Data type: ", t4.dtype,"\n")
	Tensor: tensor ([1, 2, 3, 4, 5, 6])
	Data type: torch.int64
	Tensor: tensor ([1., 2., 3., 4., 5., 6.])
	Data type: torch.float32
	Tensor: tensor([1.5000, 3.4000, 6.8000, 9.3000, 7.0000, 2.80e0], dtype=torch.float64)
	Data type: torch. float64
	Tensor: tensor ([1.5000, 3.4000, 6.8000, 9.3000, 7.0000, 2.800e], dtype=torch.float64)
	Data type: torch. float64

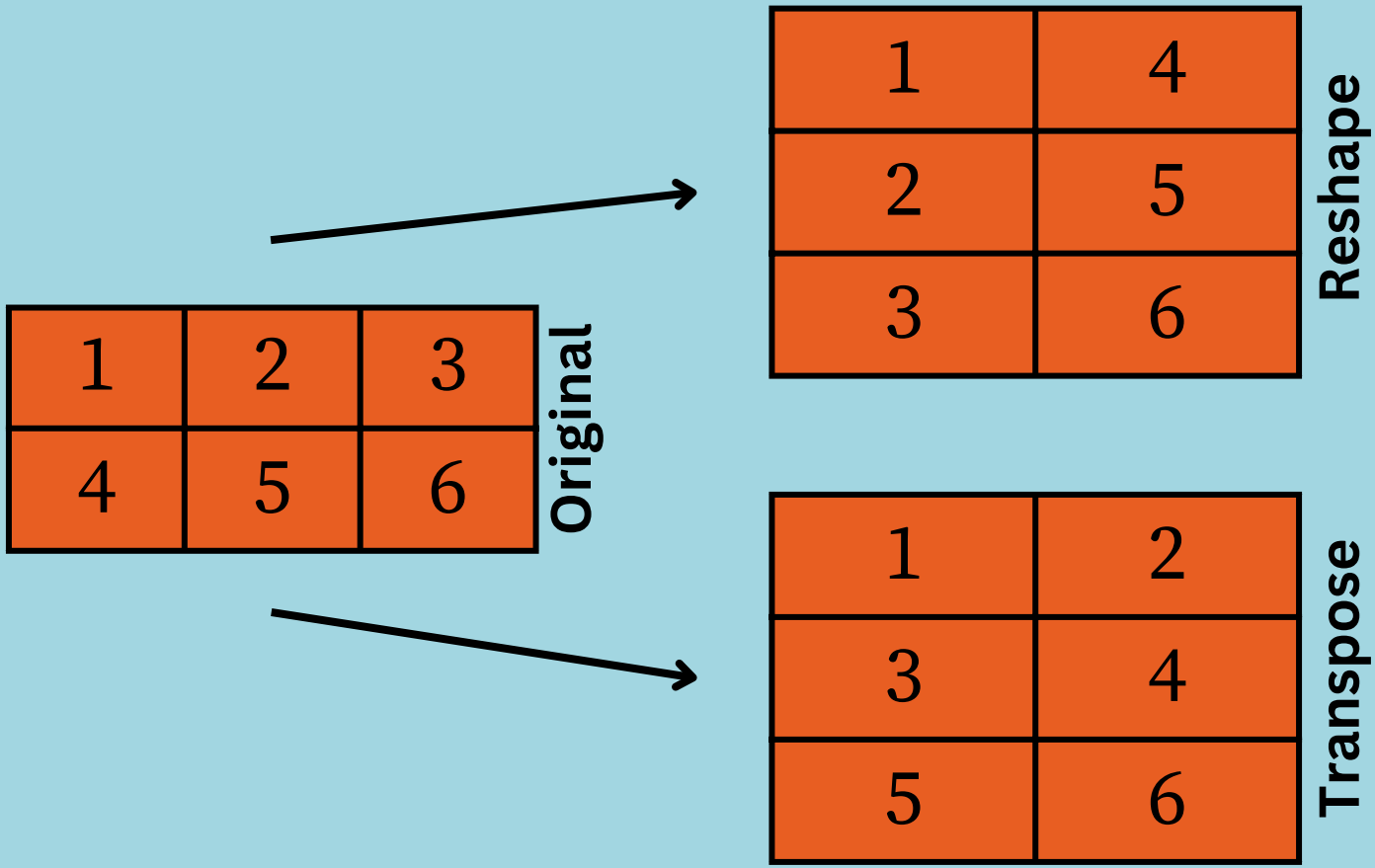
RESTRUCTURING TENSORS IN PYTORCH

We can modify the shape and size of a tensor as desired in PyTorch. We can also create a transpose of an n-d tensor. Below are three common ways to change the structure of your tensor as desired:

.reshape(a, b)
returns a new tensor with size a,b

.resize(a, b)
returns the same tensor with the size a,b

.transpose(a, b)
returns a tensor transposed in a and b dimension



Reshape and Transpose Difference

A 2*3 matrix has been reshaped and transposed to 3*2. We can visualize the change in the arrangement of the elements in the tensor in both cases.

Code	Output

# import torch module 	
import torch
 	
# defining tensor 	
t = torch.tensor([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
 	
# reshaping the tensor 	
print("Reshaping")
print(t.reshape(6, 2))
 	
# resizing the tensor 	
print("\nResizing")
print(t.resize(2, 6))
 	
# transposing the tensor 	
print("\nTransposing")
print(t.transpose(1, 0))
	Reshaping
	tensor([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
	
	Resizing
	tensor ([[1, 2, 3, 4, 5, 6], [7, 8, 9, 10, 11, 12]])
	
	Transposing
	tensor([[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]])

MATHEMATICAL OPERATIONS ON TENSORS USING PYTORCH

We can perform various mathematical operations on tensors using Pytorch. The code for performing Mathematical operations is the same as in the case with NumPy arrays. Below is the code for performing the four basic operations in tensors.

CODE

```
# import torch module
import torch
 # defining two tensors
t1 = torch.tensor([1, 2, 3, 4])
t2 = torch.tensor([5, 6, 7, 8])
 # adding two tensors
print("tensor2 + tensor1")
print(torch.add(t2, t1))
 # subtracting two tensor
print("\ntensor2 - tensor1")
print(torch.sub(t2, t1))
 # multiplying two tensors
print("\ntensor2 * tensor1")
print(torch.mul(t2, t1))
 # diving two tensors
print("\ntensor2 / tensor1")
print(torch.div(t2, t1))
```

OUTPUT

```
Tensor: tensor ([1, 2, 3, 4, 5, 6])
Data type: torch.int64
Tensor: tensor ([1., 2., 3., 4., 5., 6.])
Data type: torch.float32
Tensor: tensor([1.5000, 3.4000, 6.8000, 9.3000,
7.0000, 2.80e0], dtype=torch.float64)
Data type: torch. float64
Tensor: tensor ([1.5000, 3.4000, 6.8000, 9.3000,
7.0000, 2.800e], dtype=torch.float64)
Data type: torch. float64
```

PYTORCH MODULES

The PyTorch library modules are essential to create and train neural networks. The three main library modules are Autograd, Optim, and nn.

 # 1. Autograd Module

The autograd provides the functionality of easy calculation of gradients without the explicitly manual implementation of forward and backward pass for all layers. For training any neural network we perform backpropagation to calculate the gradient. By calling the .backward() function we can calculate every gradient from the root to the leaf.

 # 2. Optim Module

PyTorch Optium Module which helps in the implementation of various optimization algorithms. This package contains the most commonly used algorithms like Adam, SGD, and RMS-Prop. To use torch.optim we first need to construct an Optimizer object which will keep the parameters and update it accordingly. First, we define the Optimizer by providing the optimizer algorithm we want to use. We set the gradients to zero before backpropagation. Then for updation of parameters the optimizer.step() is called.

PYTORCH MODULES

 # 3. nn Module

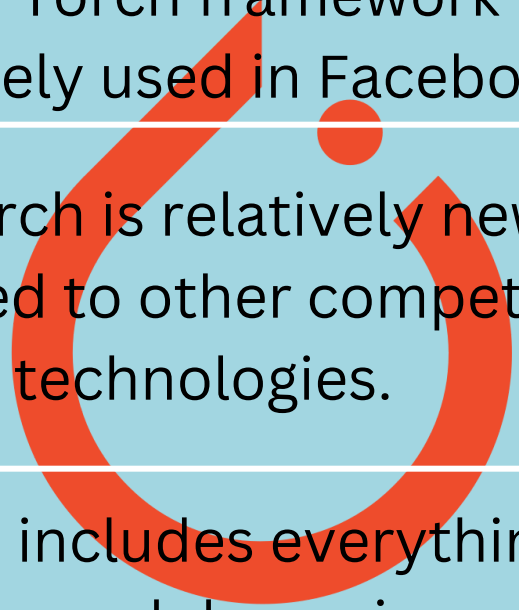
This package helps in the construction of neural networks. It is used to build layers.

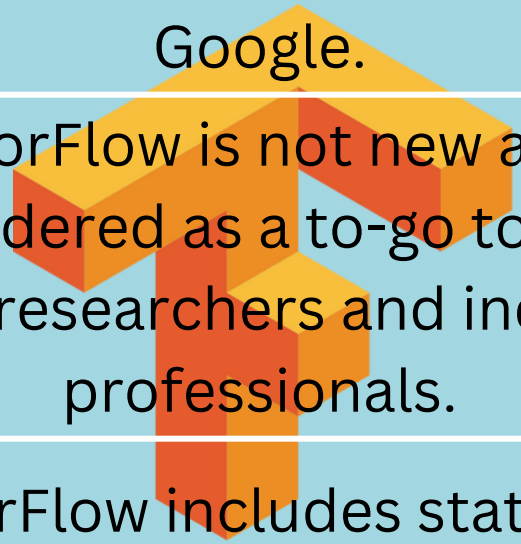
For creating a model with a single layer we can simply define it by using nn.Sequential().

For the implementation of the model which is not in a single sequence, we define a model by subclassing the nn.Module class.

PYTORCH VS TENSORFLOW

PyTorch	TensorFlow
PyTorch is closely related to the lua-based Torch framework which is actively used in Facebook.	TensorFlow is developed by Google Brain and actively used at Google.
PyTorch is relatively new compared to other competitive technologies.	TensorFlow is not new and is considered as a to-go tool by many researchers and industry professionals.
PyTorch includes everything in imperative and dynamic manner.	TensorFlow includes static and dynamic graphs as a combination.
Computation graph in PyTorch is defined during runtime.	TensorFlow do not include any run time option.





KEY FEATURES OF USING PYTORCH

TENSOR COMPUTATION

Tensors are generic n-dimensional arrays used for arbitrary numeric computation and are accelerated by graphics processing units. These multidimensional structures can be operated on and manipulated with application program interfaces (APIs).

TORCHSCRIPT

This is the production environment of PyTorch that enables users to seamlessly transition between modes. TorchScript optimizes functionality, speed, ease of use and flexibility.

AUTOMATIC DIFFERENTIATION

This technique is used for creating and training neural networks. It numerically computes the derivative of a function by making backward passes in neural networks.

KEY FEATURES OF USING PYTORCH

DYNAMIC GRAPH COMPUTATION

This feature lets users change network behavior on the fly, rather than waiting for all the code to be executed.

PYTHON SUPPORT

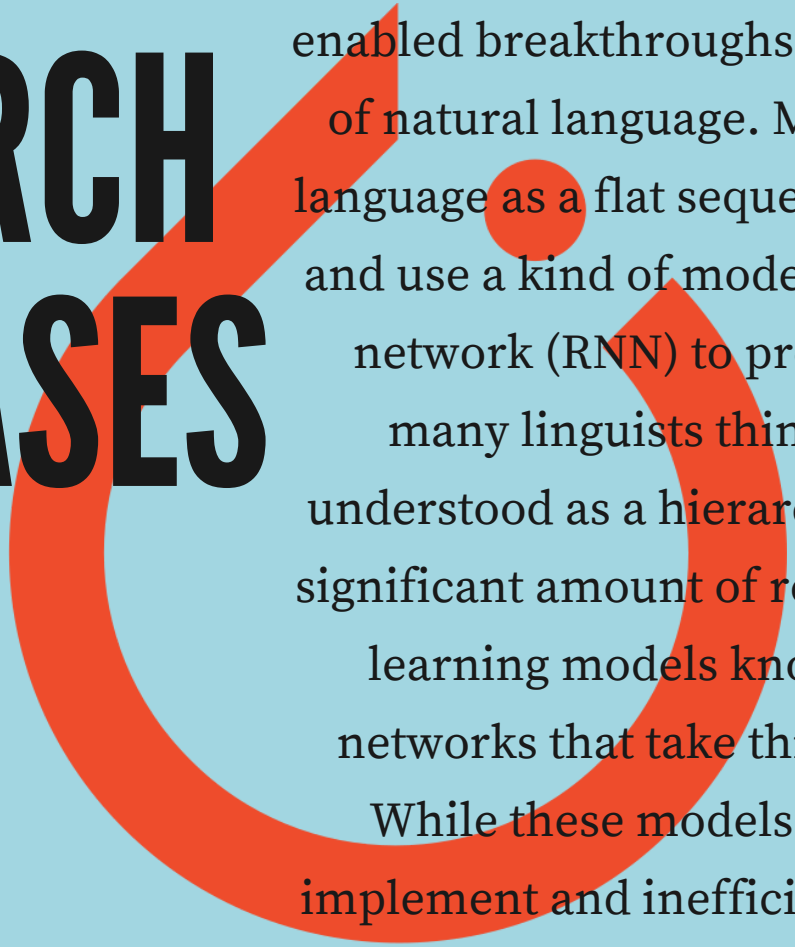
Because PyTorch is based on Python, it can be used with popular libraries and packages such as NumPy, SciPy, Numba and Cynthon.

PYTORCH BENIFITS

- Offers developers an easy-to-learn, simple-to-code structure that's based on Python.
- Enables easy debugging with popular Python tools.
- Offers scalability and is well-supported on major cloud platforms.
- Provides a small community focused on open source.
- Exports learning models to the Open Neural Network Exchange (ONNX) standard format.
- Offers a user-friendly interface.
- Provides a C++ front-end interface option.
- Includes a rich set of powerful APIs that extend the PyTorch library.

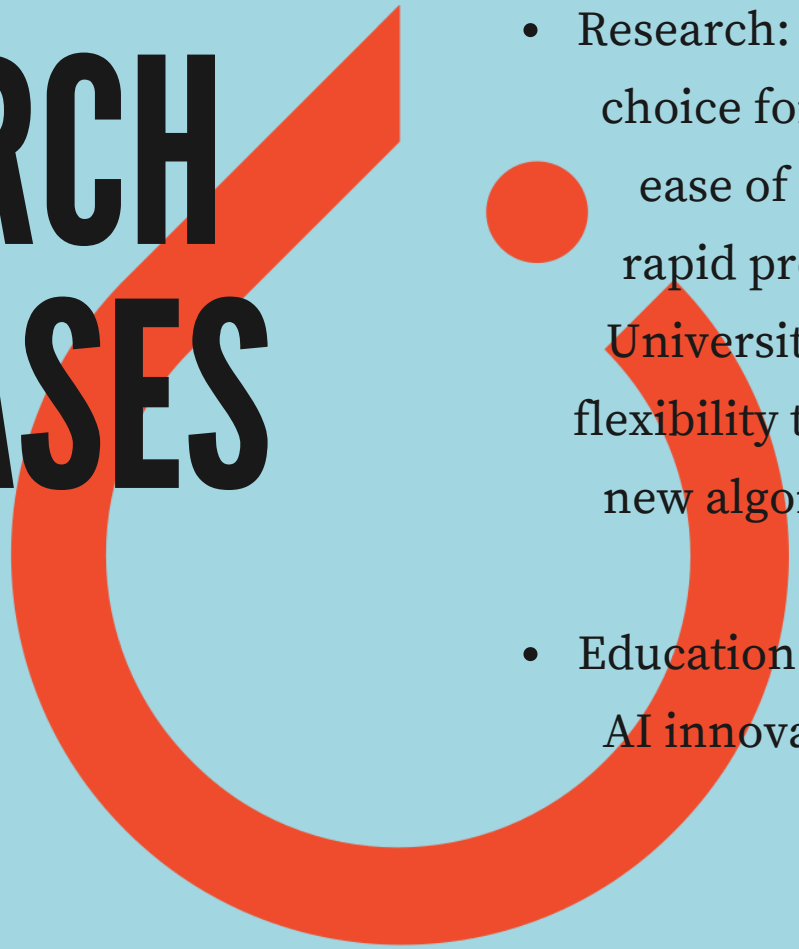
MEMBERS AND GOVERNING BOARD

PYTORCH USE CASES



- Natural Language Processing (NLP): From Siri to Google Translate, deep neural networks have enabled breakthroughs in machine understanding of natural language. Most of these models treat language as a flat sequence of words or characters and use a kind of model called a recurrent neural network (RNN) to process this sequence. But many linguists think that language is best understood as a hierarchical tree of phrases, so a significant amount of research has gone into deep learning models known as recursive neural networks that take this structure into account. While these models are notoriously hard to implement and inefficient to run, PyTorch makes these and other complex natural language processing models a lot easier. Salesforce is using PyTorch for NLP and multi-task learning.

PYTORCH USE CASES



- Research: PyTorch is a popular choice for research due to its ease of use, flexibility and rapid prototyping. Stanford University is using PyTorch's flexibility to efficiently research new algorithmic approaches.

- Education: Udacity is educating AI innovators using PyTorch.

THANKS FOR YOUR TIME!