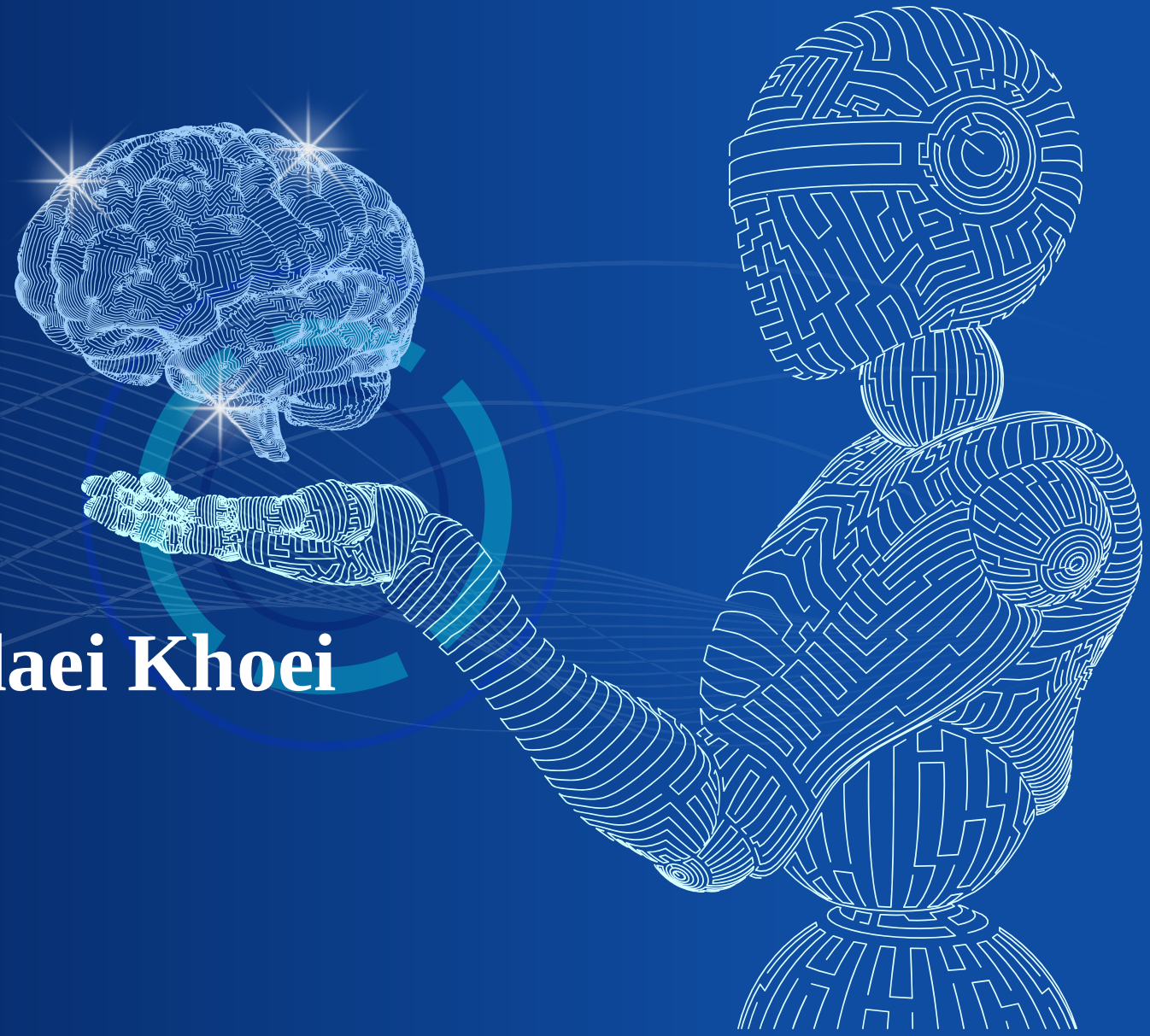
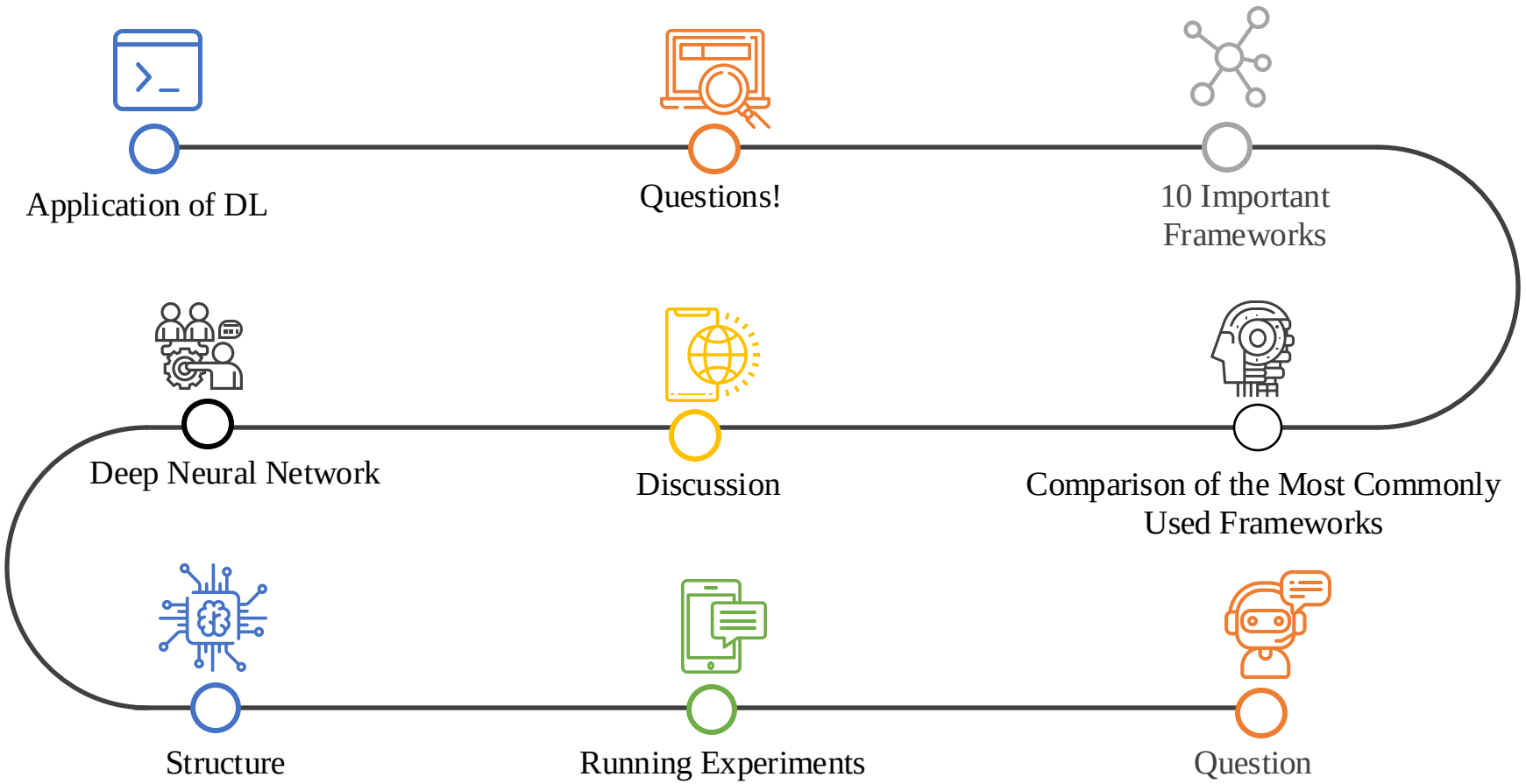
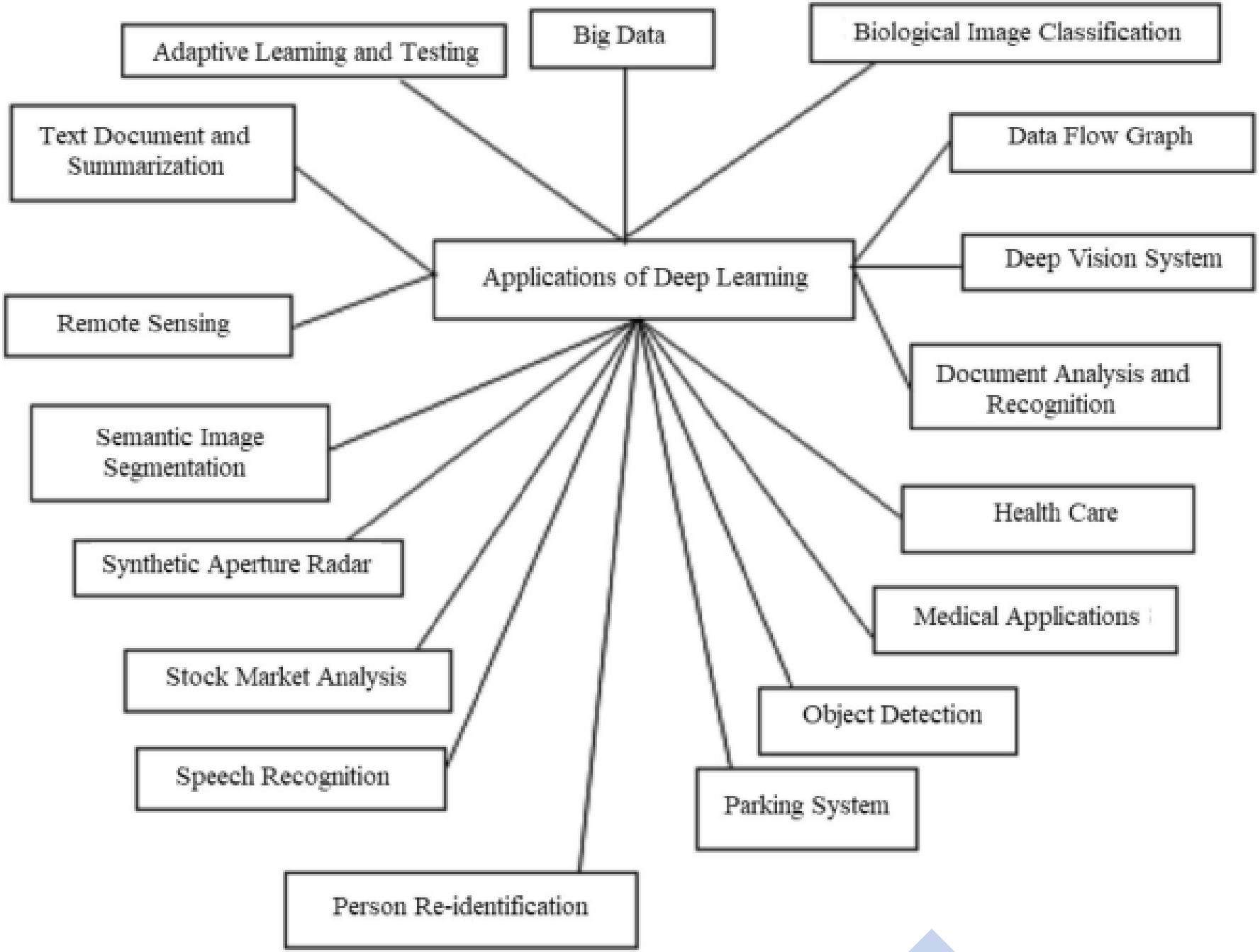
CS 7150: Deep Learning



Presented by: Dr. Tala Talaei Khoei

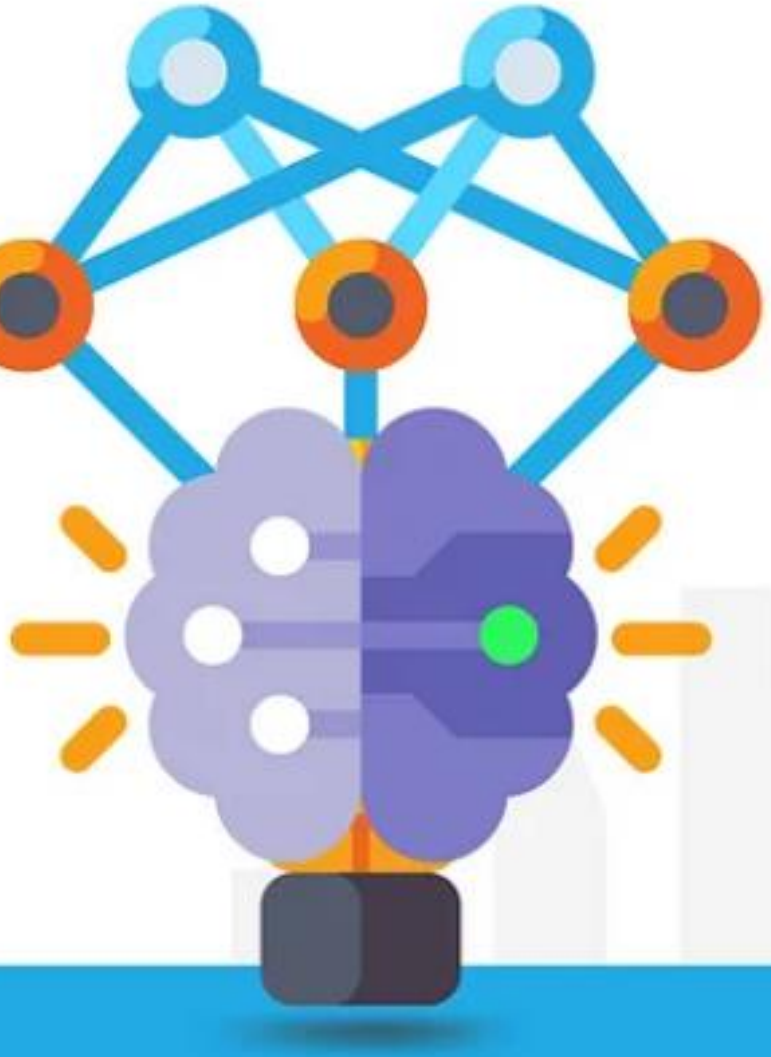
Today Outline




Applications of Deep Learning
Adaptive Learning and Testing
Big Data
Biological Image Classification
Text Document and Summarization
Data Flow Graph
Deep Vision System
Remote Sensing
Document Analysis and Recognition
Semantic Image Segmentation
Health Care
Synthetic Aperture Radar
Medical Applications
Stock Market Analysis
Object Detection
Speech Recognition
Parking System
Person Re-identification

POPULAR DEEP LEARNING FRAMEWORKS

- TensorFlow
- TORCH/PyTorch
- DEEPLEARNING4J
- CNTK
- KERAS
- ONNX
- MXNET
- Theano
- Caffe
- Chainer



TensorFlow

It is a deep learning framework example for machine learning and artificial intelligence. This is one of the popular deep learning frameworks. This deep learning framework is primarily used for the training and inference of neural networks. It supports programming languages such as Python and C++ and is compatible with various platforms such as macOS, Microsoft Windows, and Android.

Advantages:

- It supports both CPU and GPU computing devices.
- It is scalable and the process of debugging is easy. This makes it the easiest deep learning framework.

Disadvantage:

- It has extremely limited features open for Windows users.
- Comparatively, it is a slow deep-learning framework.

TORCH/PyTorch

Pytorch is the second in the list of popular deep learning frameworks. This is also an open-source deep learning framework. Pytorch is the new framework and is completely based on the Torch library. It is developed by the Facebook AI research group and open-sourced GitHub. It has two high-level features, namely Tensor computing and an automatic differentiation system.

Advantages:

•It is flexible and has efficient memory usage.
•It makes coding more manageable.
•The process of debugging is easy.

Disadvantage:

•The developer community is small in this framework as compared to other deep learning platforms.

DEEPLEARNING4J

It is a framework written in JAVA and gives support to deep learning algorithms. This deep learning framework also supports various Java virtual machine-based programming languages such as Scala, Kotlin, and many more.

Advantages:

- This framework is easily scalable.
- It has easy integration with Apache Spark.

Disadvantage:

- It is only limited to Java programming language.

THE MICROSOFT COGNITIVE TOOLKIT/CNTK

This deep learning framework is used to train deep learning neural network models. It also helps to train text formatted data and images. It has features that make this framework highly efficient and scalable for various machines. This framework is supported by interfaces such as Command Line, Python, and C++.

Advantages:

•RNN and CNN neural networks are supported by this deep learning framework.

Disadvantage:

•It has minimal community support compared to other deep learning frameworks.

KERAS

KERAS framework comes fifth in the list of deep learning frameworks. It is an open-source, high-level application programable interface development by Google. You can develop deep-learning neural networks using this API. With the help of this API, we can decrease the number of user actions that are required for common use cases. It is the most common framework used by NASA and CERN.

Advantages:

It offers a simple API.
It is easy to learn and use.
It is user-friendly and provides fast deployment of neural networks.

Disadvantage:

The process of debugging is difficult in this framework.
Data preprocessing tools that come with this framework are not that good compared to other framework tools.

ONNX

ONNX (Open neural network Exchange) comes next in the deep learning framework list. It is an open-source AI ecosystem of technology companies and research organizations**.** This ONNX was created by Facebook and Microsoft.

It has a common set of operators that helps in building deep learning and machine learning models. This framework helps in maximizing the interoperability and flexibility between different models.

Advantages:

- It has C, JAVA, python, and C# API. These APIs can be used in various environments.
- This framework works well for MAC, Linux, and Windows.
- It helps in enhancing the compatibility between various other frameworks.

Disadvantage:

- It has some new features that may raise a sense of doubt among a few users

MXNET

It is also an open-source deep learning framework given by Apache. MXNET allows you to create, train and deploy neural networks. This framework supports different programming languages such as Julia, Scala, Java, C++, R, and Perl, etc.
Some of its tool kits are Gluon TS and GluonCV.

Advantages:
This framework is fast, scalable, and efficient.
APACHE MXNET is supported by all the major platforms
Disadvantage:
Compared to other major frameworks such as TensorFlow, it has a smaller open-source community.
Due to a lack of community support, it takes a little long sometimes to fix the bug.

Theano

It is a deep learning framework based on a deep learning library. Theano is compatible with both CPU and GPU. This framework was written in Python and CUDA programming languages. This framework is compatible with platforms such as Linux, macOS, and Windows.

Advantages:

•It is an open-source deep learning framework.

•It contains high-level wrappers, namely Kera and Lasagne that help in increasing its usability.

Disadvantage:

•It is not very compatible with AWS (Amazon Web Services).

•Large in-size models need more compilation times in this deep learning framework.

Caffe

Convolutional Architecture for Fast Feature Embedding is a low-level library developed in California. This framework is well suited for CNN, which allows easy switching from GPU and CPU. In this framework multi-GPU, tanning is partially supported although it is aimed at edge deployment.

Advantages:

•It is flexible in some of its features.
•This framework is fast.

Disadvantage:

•It is hard to use compared to its competitors.

Chainer

It is an open-source deep learning framework. It is completely written in Python programming language. This deep learning framework supports different architectures including convnets, feed-forward nets, recursive nets, and recurrent. It also supports per-batch architectures.

Advantages:

- It runs multiple GPUs in little time.
- One of the powerful and flexible deep learning frameworks.

Disadvantage:

- The community is small.
- It has a difficult debugging process.

Overall Comparison

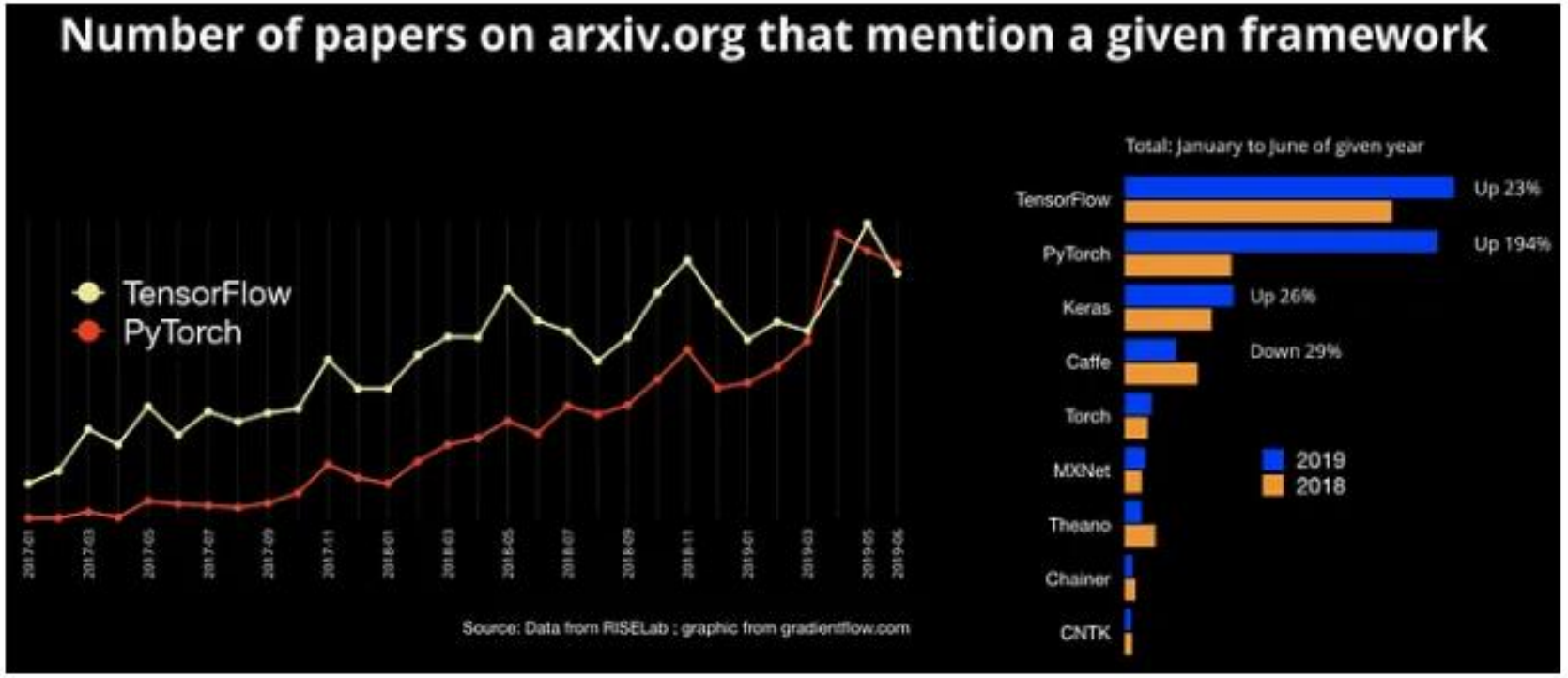
The two frameworks that are the most popular (and for good reasons) are TensorFlow/Keras and PyTorch. Overall, for deep learning applications in general, these are arguably the best frameworks to use. Both frameworks offer a balance between high-level APIs and the ability to customize your deep learning models without compromising on functionality. I am personally a fan of Keras and if I had to choose between PyTorch and Keras I would choose Keras as the best overall deep learning framework. However, PyTorch is definitely not far behind and MXNet is also a decent option to consider.

Best for Beginners

Keras is easily the best framework for beginners to start with. The API is extremely simple and easy to understand. There is a reason why François Chollet, the creator of Keras, used the phrase “deep learning for humans” to describe his library. If you are a beginner just getting started with deep learning or even an experienced deep learning practitioner who wants to quickly put together a deep learning model in just a few lines of code, I would recommend starting with Keras.

Best for Advanced Research

In advanced research involving deep learning, especially where the goal is to design new architectures or come up with novel methods, the ability to improvise and create highly customized neural networks is extremely important. For this reason, PyTorch, which offers a large ecosystem of additional libraries that support it as well as the ability to extend and customize its modules, is gaining popularity in research. Consider the graph below in which PyTorch achieved a 194 percent increase in arxiv.org paper mentions in the given period from 2018 to 2019. Compare this statistic to the 23 percent increase in mentions for TensorFlow and it is clear that PyTorch is growing faster in the research community. In fact, this data is a year old and PyTorch may have already surpassed TensorFlow in the research community as of 2020.



Summary

- TensorFlow/Keras and PyTorch are overall the most popular and arguably the two best frameworks for deep learning.
- If you are a beginner who is new to deep learning, Keras is probably the best framework for you to start out with.
- If you are a researcher looking to create highly-customized architectures, you might be slightly better off choosing PyTorch over TensorFlow/Keras.

Compare Three Most Popular Deep Learning Framework (TensorFlow, PyTorch, Keras)

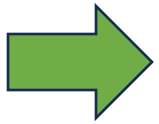
	TensorFlow	PyTorch	Keras
API Level	Both (High and Low)	Low	High
Architecture	Not easy to use	Complex, less readable	Simple, concise, readable
Datasets	Large datasets, high-performance	Large datasets, high-performance	Smaller datasets
Debugging	Difficult to conduct debugging	Good debugging capabilities	Simple network, so debugging is not often needed
Pretrained models?	Yes	Yes	Yes
Popularity	Second most popular of the three	Third most popular of the three	Most popular of the three
Speed	Fast, high-performance	Fast, high-performance	Slow, low performance
Written In	C++, CUDA, Python	Lua	Python

Simple. Flexible. Powerful.

Keras Introduction

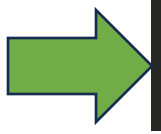
- Keras is a high-level, user-friendly API used for building and training neural networks. It is an open-source library built in Python that runs on top of TensorFlow. It was developed to enable fast experimentation and iteration, and it lowers the barrier to entry for working with deep learning.

Step 1: *Installing Keras*



```
# First upgrade pip
pip install --upgrade pip

# Then install TensorFlow
pip install tensorflow
```

Step 2



```
from tensorflow import keras
```

Step 3: *Building a Model with Keras*

```
from tf.keras import Sequential

model = Sequential()
```

Step 4

```
from tf.keras import Dense

model.add(Dense(input_shape=(16,))
model.add(Dense(8))
model.add(Dense(4))
model.add(Dense(2))
model.add(Dense(1))
```

Step 5: *Compiling a Model with Keras*

```
model.compile(loss='mean_squared_error', optimizer='adam')
```

Step 6: *Fitting a Model with Keras*

```
model.fit(X_train, y_train)
```

Step 7

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
```


PyTorch

PyTorch
PyTorch is a powerful, yet easy-to-use deep learning library for Python, mainly used for applications such as computer vision and natural language processing.

Tensors: In deep learning, tensors are a fundamental data structure that is very similar to arrays and matrices, with which we can efficiently perform mathematical operations on large sets of data. A tensor can be represented as a matrix, but also as a vector, a scalar, or a higher-dimensional array.

```
# Upgrade pip
pip install --upgrade pip

# Install the current stable release of tensorflow.
pip install tensorflow
```

```
import torch
import numpy as np

ndarray = np.array([0, 1, 2])
t = torch.from_numpy(ndarray)
print(t)
```

A tensor on PyTorch has three attributes:

- **shape**: the size of the tensor
- **data type**: the type of data stored in the tensor
- **device**: the device in which the tensor is stored

```
print(t.shape)
print(t.dtype)
print(t.device)
```

Tensor Operations

Just like in NumPy, there are multiple possible operations we can perform with tensors--like slicing, transposing, and multiplying matrices, among others.

The slicing of a tensor is done exactly like any other array structure in Python. Consider the tensor below:

```
zeros_tensor = torch.zeros((2, 3))
print(zeros_tensor)
```

Finally, we can multiply the tensors:

```
ones_tensor = torch.ones(3, 3)
product = torch.matmul(zeros_tensor, ones_tensor)
print(product)
```

Loading Data

PyTorch comes with a built-in module that provides ready-to-use datasets for many deep learning applications, such as computer vision, speech recognition, and natural language processing. This means that it's possible to build your own neural network without the need to collect and process data yourself.

```
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt

training_data = datasets.MNIST(root=".", train=True, download=True, transform=ToTensor())

test_data = datasets.MNIST(root=".", train=False, download=True, transform=ToTensor())
```

Neural Networks

In deep learning, a neural network is a type of algorithm used to model data with complex patterns. A neural network attempts to simulate the functioning of the human brain through multiple layers connected by processing nodes, which behave like human neurons. These layers connected by nodes create a complex net that is able to process and understand huge amounts of complex data.

In PyTorch, everything related to neural networks is built using the torch.nn module. The network itself is written as a class that inherits from nn.Module, and, inside the class, we'll use nn to build the layers.

- The nn.Flaten is responsible for transforming the data from multidimensional to one dimension only.
- The nn.Sequential container creates a sequence of layers inside the network.

```python
from torch import nn

class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

```
model = NeuralNetwork()
print(model)
```

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

Training the Neural Network

Now that we have defined our neural network, we can put it to use. Before starting the training, we should first set a loss function. The loss function measures how far our model is from the correct results, and it's what we'll try to minimize during the training of the network. Cross-entropy is a common loss function used for classification tasks, and it's the one we'll use. We should initialize the function:

```
loss_function = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
```

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        pred = model(X)
        loss = loss_fn(pred, y)

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

        if batch
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

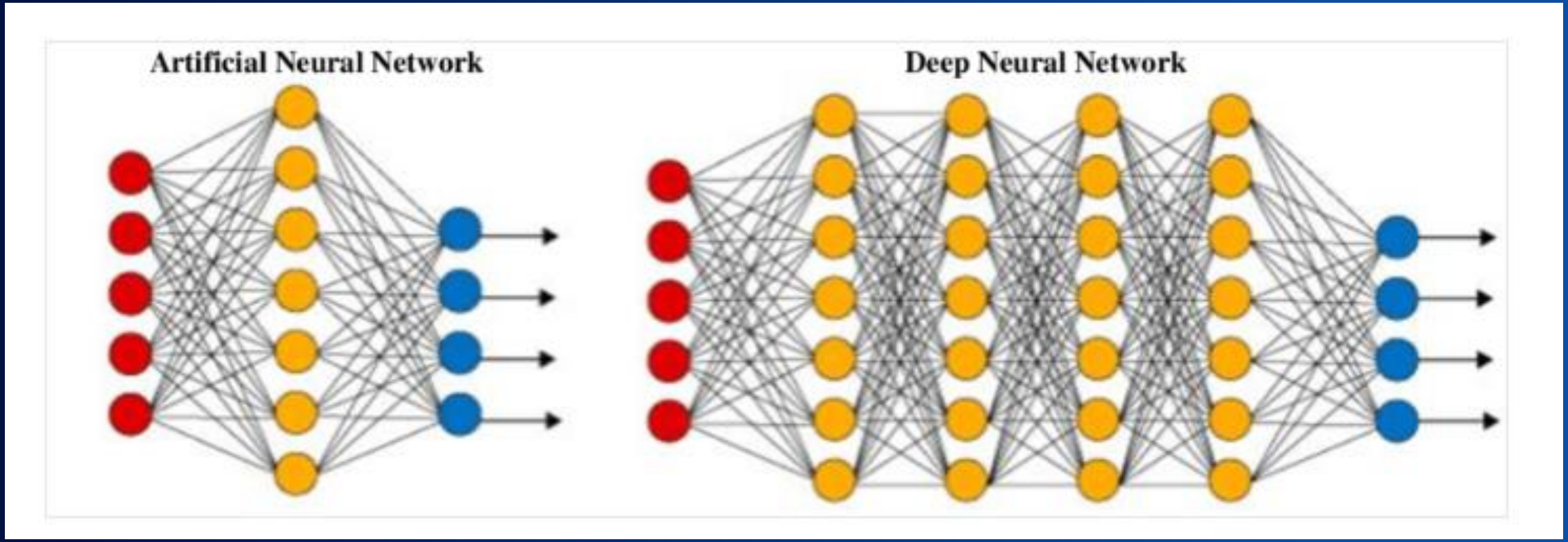
```
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f
We then set the number of epochs to train our model. An epoch consists of an iteration ove
This is PyTorch's implementation and the output of such a loop:
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(loaded_train, model, loss_function, optimizer)
    test(loaded_test, model, loss_function)
print("Done!")
```

Saving the Model

```
Notice that in each epoch, we print the loss function at every 100 batches in the training loop,
If we had set more epochs--let's say 10, 50, or even 100--chances are we'd see even better resul
With our model finally trained, it's easy to save it and load it when necessary:
torch.save(model, "model.pth")
model = torch.load("model.pth")
```

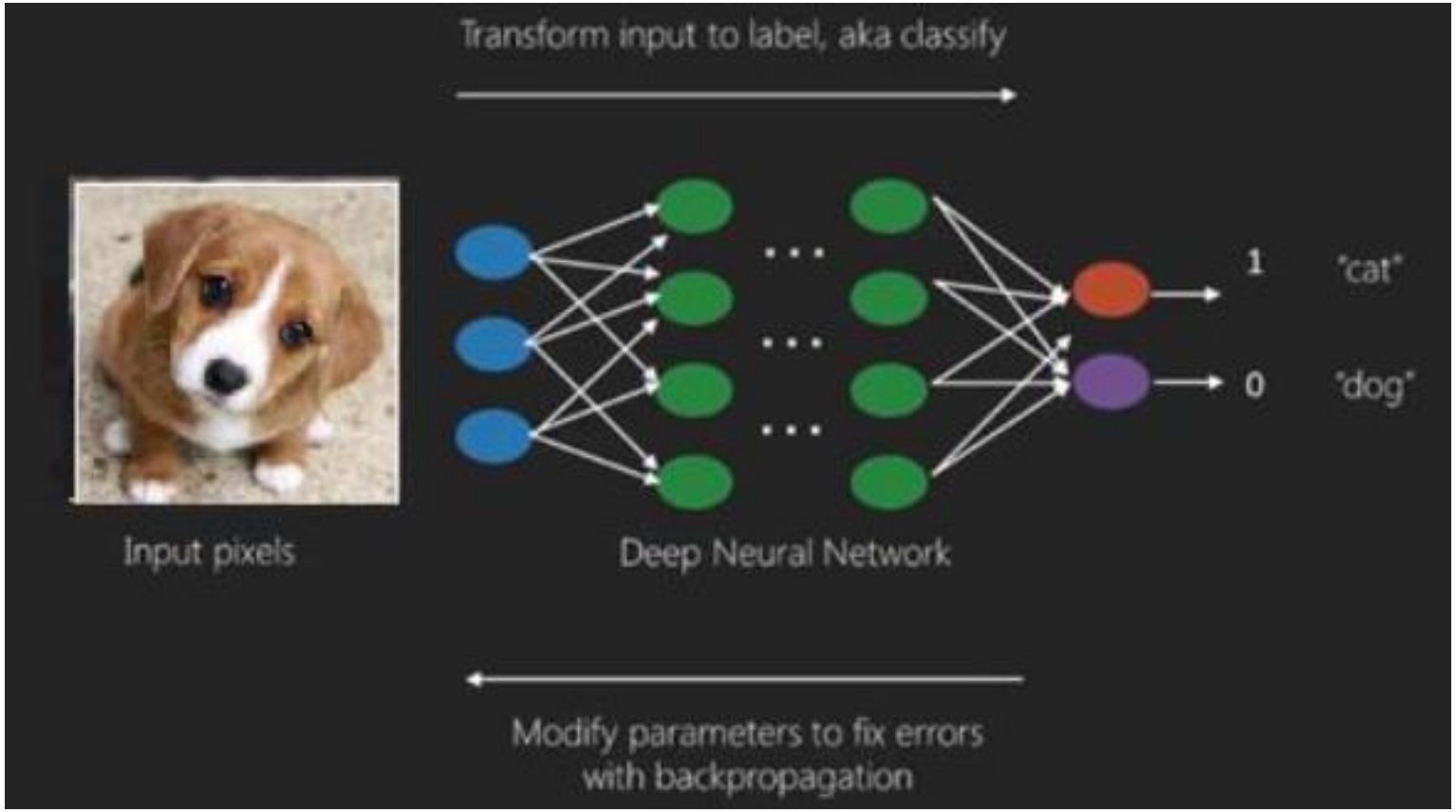
Please Refer to YouTube Videos for the TensorFlow, Keras, and PyTorch Frameworks

Deep Neural Networks



Deep Neural Network

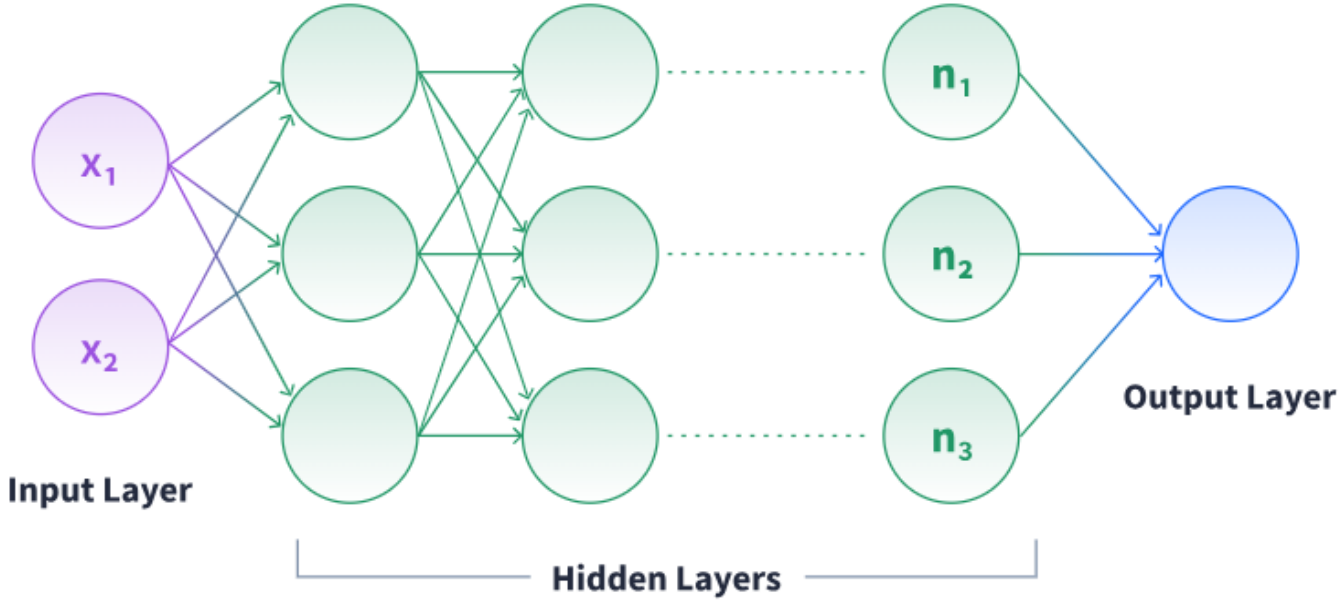
- A deep neural network (DNN) is an ANN with multiple hidden layers between the input and output layers. Similar to shallow ANNs, DNNs can model complex non-linear relationships.
- The main purpose of a neural network is to receive a set of inputs, perform progressively complex calculations on them, and give output to solve real world problems like classification. We restrict ourselves to feed forward neural networks.



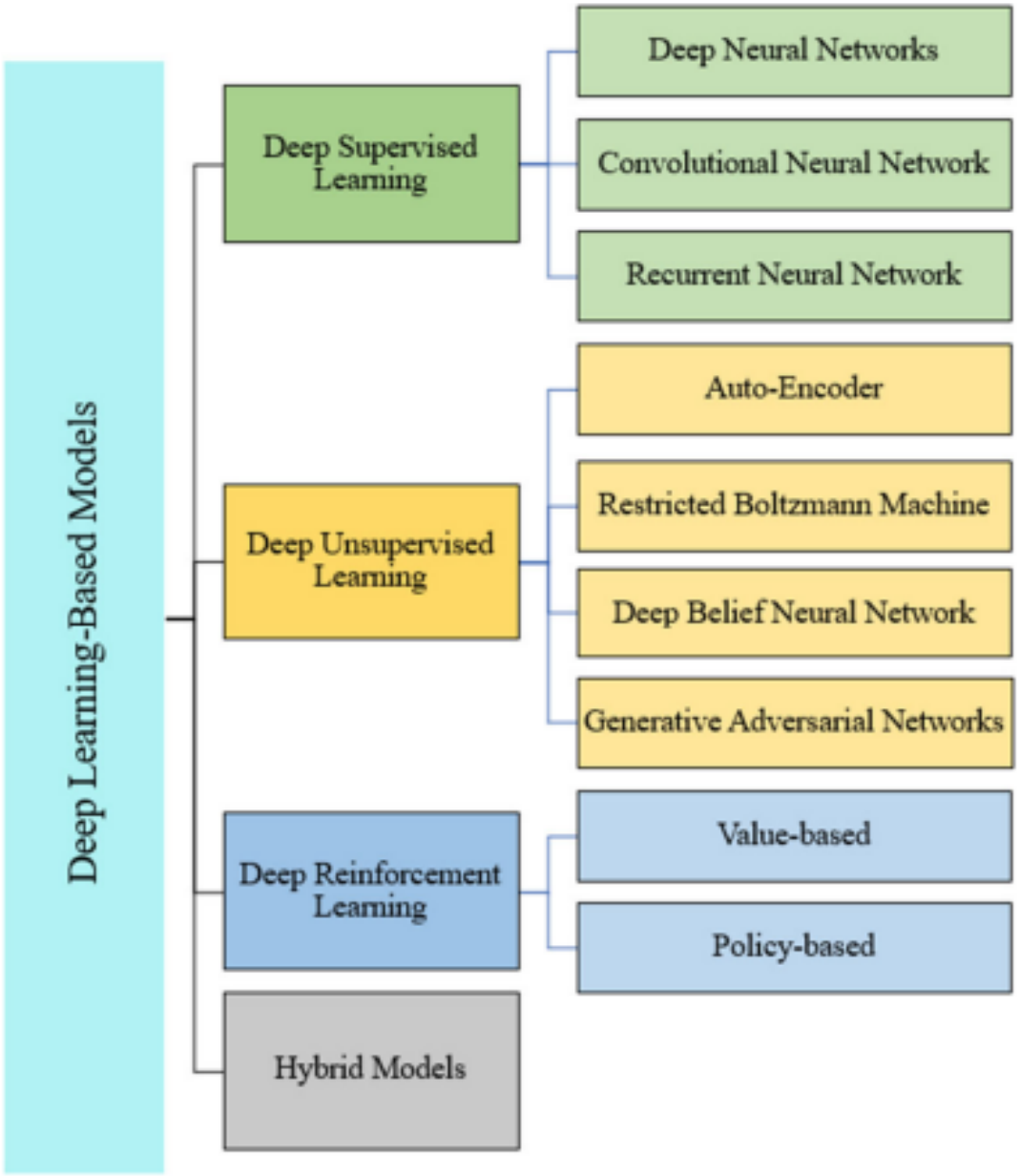
Structure:

- A **deep (dense) neural network** consist of multiple hidden layers. Each layer contains a set of neurons that learn to extract certain features from the data. The output layer produces the final results of the network. The image below represents the basic architecture of a deep neural network with n-hidden layers.

- The additional hidden layers in a deep neural network enable it to learn more complex patterns than a shallow neural network. Consequently, deep neural networks are more accurate but also more computationally expensive to train than shallow neural networks. Therefore, deep neural networks are preferable for complex, real-time, real-world applications such as multivariate time series forecasting, natural language processing, real-time forecasting, or predictive lead times.



Neural Network Models



Note: We will cover these Models Architecture in next sessions

How to Train and Model a Neural Network?

1. Model the input layer according to the no. of input features.
2. Model the output layer according to the no. of classes in the output.
3. Model the number of hidden layers and the no. of neurons in the hidden layers optimally.
4. Randomly initialize weights.
5. Implement forward propagation for any x.
6. Implement code to compute cost function .
7. Implement backpropagation to compute the gradient of cost with respect to weights.
8. Update the values of weights.
9. Perform steps five-through-nine recursively to minimize the cost by modifying the weights after each epoch.

Please Refer to Examples of DNN on Canvas


THANK YOU
Any Questions?