Keras

Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.

It cannot handle low-level computations, so it makes use of the **Backend** library to resolve it. The backend library act as a high-level API wrapper for the low-level API, which lets it run on TensorFlow, CNTK, or Theano.

Keras special

- Focus on user experience has always been a major part of Keras.
- Large adoption in the industry.
- It is a multi backend and supports multi-platform, which helps all the encoders come together for coding.
- Research community present for Keras works amazingly with the production community.
- Easy to grasp all concepts.
- It supports fast prototyping.
- It seamlessly runs on CPU as well as GPU.
- It provides the freedom to design any architecture, which then later is utilized as an API for the project.
- It is really very simple to get started with.
- Easy production of models actually makes Keras special.

user experience

1. **Keras is an API designed for humans**
   Best practices are followed by Keras to decrease cognitive load, ensures that the models are consistent, and the corresponding APIs are simple.

2. **Not designed for machines**
   Keras provides clear feedback upon the occurrence of any error that minimizes the number of user actions for the majority of the common use cases.
3. **Easy to learn and use.**
4. **Highly Flexible**
   Keras provide high flexibility to all of its developers by integrating low-level deep learning languages such as TensorFlow or Theano, which ensures that anything written in the base language can be implemented in Keras.

Keras support the claim of being multi-backend and multi-platform

Keras can be developed in R as well as Python, such that the code can be run with TensorFlow, Theano, CNTK, or MXNet as per the requirement. Keras can be run on CPU, NVIDIA GPU, AMD GPU, TPU, etc. It ensures that producing models with Keras is really simple as it totally supports to run with TensorFlow serving, GPU acceleration (WebKeras, Keras.js), Android (TF, TF Lite), iOS (Native CoreML) and Raspberry Pi.

Keras Backend

Keras being a model-level library helps in developing deep learning models by offering high-level building blocks. All the low-level computations such as products of Tensor, convolutions, etc. are not handled by Keras itself, rather they depend on a specialized tensor manipulation library that is well optimized to serve as a backend engine. Keras has managed it so perfectly that instead of incorporating one single library of tensor and performing operations related to that particular library, it offers plugging of different backend engines into Keras.

Keras consist of three backend engines, which are as follows:

- **TensorFlow**
  TensorFlow is a Google product, which is one of the most famous deep learning tools widely used in the research area of machine learning and deep neural network. It came into the market on 9th November 2015 under the Apache License 2.0. It is built in such a way that it can easily run on multiple CPUs and GPUs as well as on mobile operating systems. It consists of various wrappers in

distinct languages such as Java, C++, or Python.

- **Theano**

  Theano was developed at the University of Montreal, Quebec, Canada, by the MILA group. It is an open-source python library that is widely used for performing mathematical operations on multi-dimensional arrays by incorporating scipy and numpy. It utilizes GPUs for faster computation and efficiently computes the gradients by building symbolic graphs automatically. It has come out to be very suitable for unstable expressions, as it first observes them numerically and then computes them with more stable algorithms.

CNTK

Microsoft Cognitive Toolkit is deep learning's open-source framework. It consists of all the basic building blocks, which are required to form a neural network. The models are trained using C++ or Python, but it incorporates C# or Java to load the model for making predictions.

Advantages of Keras

- It is very easy to understand and incorporate the faster deployment of network models.
- It has huge community support in the market as most of the AI companies are keen on using it.
- It supports multi backend, which means you can use any one of them among TensorFlow, CNTK, and Theano with Keras as a backend according to your requirement.

Disadvantages of Keras

- The only disadvantage is that Keras has its own pre-configured layers, and if you want to create an abstract layer, it won't let you because it cannot handle low-level APIs. It only supports high-level API running on the top of the backend engine (TensorFlow, Theano, and CNTK).

Keras is a model-level library, offers high-level building blocks that are useful to develop deep learning models. Instead of supporting low-level operations such as tensor products, convolutions, etc. itself, it depends upon the backend engine that is well specialized and optimized tensor manipulation library. It doesn't pick just one library of a tensor to implement Keras tied to that particular library. It handles the situation in a modular way by seamlessly plugging many distinct back-end engines to Keras.

Keras backends

Following are the three available backend implementations, which are as follows;

- **TensorFlow:** This Google-developed framework for symbolic tensor manipulation is open-source.
- **Theano:** It is also an open-source framework for symbolic manipulation of a tensor is developed at Universite de Montreal by LISA Lab.
- **CNTK:** It is developed by Microsoft, which is also an open-source deep-learning toolkit.

Keras layers

Keras Core layer comprises of a **dense** layer, which is a dot product plus bias, an **activation** layer that transfers a function or neuron shape, a **dropout** layer, which randomly at each training update, sets a fraction of input unit to zero so as to avoid the issue of overfitting, a **lambda** layer that wraps an arbitrary expression just like an object of a Layer, etc.

The Keras convolution layer utilizes filters for the creation of a feature map, runs from 1D to 3D. It includes most of the common invariants, for example, **cropping** and **transposed convolution** layer for each dimension. The **2D convolution** is used for image recognition as it is inspired by the visual cortex.

The Model class

```
tf.keras.Model()
```

It is very beneficial in alliancing the layers into an object that encompasses features like training and inference.

Arguments

- **inputs:** It can be defined as an input that is being fed to the model. It can either be an object of **Input** or a list of objects, i.e., keras.Input.
- **outputs:** It refers to the model's output.
- **name:** It can be a string that defines the model's name.

Sequential class

The Keras **sequential** class helps to form a cluster of a layer that is linearly stacked into tf.keras.Model. The features of training and inference are provided by sequential to this model.

add Method

1. Sequential.add(layer)

With the help of the add method, you will be permitted to add the layer's instance located at the top of the stack layer.

Arguments

- **layer:** It can be defined as an instance of a layer.

Raises

- **TypeError:** The **TypeError** is generated in case if the **layer** is not an instance of a layer.
- **ValueError:** If the argument called **layer** is not aware of the shape of the input, then the **ValueError** is generated.
- **ValueError:** In case if the layer argument encompasses several tensor output or simply, we can say it is connected anywhere like forbidden in the Sequential model.

pop Method

1. Sequential.pop()

It helps in the removal of the last layer from the model.

Raises

- **TypeError:** This type of error is generated if it consists of a single layer within the model.

New) Arguments

- **units:** It refers to a positive integer that acknowledges the output space dimensionality.
- **activation:** It makes sure that the dense layer utilizes the element-wise activation function. It is a linear activation, which is set to none by default. Since its linearity is limited, we don't have much of its in-built activation function.
- **use_bias:** It is an optional parameter, which means we may or may not incorporate it in our calculation. It represents a Boolean that shows whether the layer utilizes a bias vector.
- **kernel_initializer:** It can be defined as an initializer for the **kernel** weights matrix.
- **bias_initializer:** It can be defined as an initializer for the bias vector for which Keras uses zero initializer by default. It is assumed that it sets the bias vector to all zeros.

- **kernel_regularizer:** It can be termed as a regularizer function, which is implemented on the **kernel** weights matrix.
- **bias_regularizer:** It can be defined as a regularizer function, which is applied to the bias vector.
- **activity_regualrizer:** It relates to a regularizer function that is applied to the output of the layer (its activation).
- **kernel_constraint:** It refers to the constraint, which is applied to the kernel weights matrix.
- **bias_constraint:** It can be defined as a constraint, which is applied to the bias vector.

Input shape

The input shape layer accepts an **nD** tensor of shape **(batch_size, ..., input_dim)**, and makes sure that its most common situation would have to be a **2D** input encompassing a shape of **(batch_size, input_dim)**.

Output shape

It outputs an **nD** tensor of shape **(batch_size, ..., units)**. For instance, where **input** is a 2D of shape **(batch_size, input_dim)**, the corresponding **output** will be of shape **(batch_size, units)**.