Karan Choudhary

Roll No:07
B1-Deep Learning
EXP:01

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.

Initially, it had over 4800 contributors during its launch, which now has gone up to 250,000 developers. It has a 2X growth ever since every year it has grown. Big companies like Microsoft, Google, NVIDIA, and Amazon have actively contributed to the development of Keras. It has an amazing industry interaction, and it is used in the development of popular firms likes Netflix, Uber, Google, Expedia, etc.

Methods in Keras:

Import Keras:

Start by importing the necessary modules from the Keras library. Typically, you'll need to import `keras` and the specific `sub-modules or classes you require, such as Sequential from keras.models and Dense from keras.layers.`

```
from keras.models import Sequential
```

```
from keras.layers import Dense
```

Create a Sequential model:

Use the `Sequential` class to create a linear stack of layers. This is the simplest way to build a
model in Keras.

```
model = Sequential()
```

Add layers:

Add layers to the model using the `add()` method. Specify the type of layer you want to add (e.g., `Dense`, `Conv2D`, etc.), along with the desired number of units/neurons and activation function.

```
model.add(Dense(units=64, activation='relu', input_dim=100))
```

```
model.add(Dense(units=10, activation='softmax'))
```

Compile the model:

Compile the model using the `compile()` method. Specify the loss function, optimizer, and metrics to be used during training.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Train the model:

Train the model using the `fit()` method. Provide the training data (`X_train`, `y_train`), specify the batch size, number of epochs, and any additional parameters required.

```
model.fit(X_train, y_train, batch_size=32, epochs=10)
```

Evaluate the model:

Evaluate the trained model on test data using the `evaluate()` method. Provide the test data (`X_test`, `y_test`) and observe the performance metrics.

```
loss, accuracy = model.evaluate(X_test, y_test)
```

```
print('Test loss:', loss)
```

```
print('Test accuracy:', accuracy)
```

Make predictions:

Use the trained model to make predictions on new/unseen data using the `predict()` method.
Provide the input data and obtain the model's predictions.

```
predictions = model.predict(X_new)
```

What is TensorFlow?

TensorFlow is a popular framework of **machine learning** and **deep learning**. It is **a free** and **open-source** library which is released on **9 November 2015** and developed by **Google Brain Team**. It is entirely based on Python programming language and use for numerical computation and data flow, which makes machine learning faster and easier.

TensorFlow can train and run the deep neural networks for image recognition, handwritten digit classification, recurrent neural network, **word embedding**, **natural language processing**, video detection, and many more. TensorFlow is run on multiple **CPU**s or **GPU**s and also mobile operating systems.

The word TensorFlow is made by two words, i.e., Tensor and Flow

- **Tensor** is a multidimensional array
- **Flow** is used to define the flow of data in operation.

TensorFlow is used to define the flow of data in operation on a multidimensional array or Tensor.

Operations in TensorFlow

Tensor Addition

You can add two tensors using `tensorA.add(tensorB)`:

```
const tensorA = tf.tensor([[1, 2], [3, 4], [5, 6]]);

const tensorB = tf.tensor([[1,-1], [2,-2], [3,-3]]);

// Tensor Addition

const tensorNew = tensorA.add(tensorB);

// Result: [ [2, 1], [5, 2], [8, 3] ]
```

Tensor Subtraction:

You can subtract two tensors using `tensorA.sub(tensorB)`:

```
const tensorA = tf.tensor([[1, 2], [3, 4], [5, 6]]);
const tensorB = tf.tensor([[1,-1], [2,-2], [3,-3]]);

// Tensor Subtraction
const tensorNew = tensorA.sub(tensorB);

// Result: [ [0, 3], [1, 6], [2, 9] ]
```

Tensor Multiplication:

You can multiply two tensors using `tensorA.mul(tensorB)`:

```
const tensorA = tf.tensor([1, 2, 3, 4]);
const tensorB = tf.tensor([4, 4, 2, 2]);

// Tensor Multiplication
const tensorNew = tensorA.mul(tensorB);

// Result: [ 4, 8, 6, 8 ]
```

Tensor Division:

You can divide two tensors using `tensorA.div(tensorB)`:

```
const tensorA = tf.tensor([2, 4, 6, 8]);
const tensorB = tf.tensor([1, 2, 2, 2]);

// Tensor Division
const tensorNew = tensorA.div(tensorB);

// Result: [ 2, 2, 3, 4 ]
```