Deep learning :-
Keras:-
Tensorflow:-

KERAS MODELS:-

*clean and convenient way to create a range of deep learning models. Keras has become one of the most used high-level neural networks APIs when it comes to developing and testing neural networks.
*Creating layers for neural networks as well as setting up complex architectures are now a breeze due to the Keras high-level API.
*A Keras model is made up of a sequence or a functional graph.
*There are several fully configurable modules that can be combined to create new models.
*Some of these configurable modules that you can plug together are neural layers, cost functions, optimizers, initialization schemes, dropout, loss, activation functions, and regularization schemes.therefore we can add all the futeres in one model.
*One of the main advantages that come with modularity is that you can easily add new features as separate modules. As a result, Keras is very flexible and well-suited for innovative research.

There are two ways you can develop a Keras model: SEQUENTIAL AND FUNCTIONAL:-
Sequential API Mode:-
The Sequential API model is the simplest model and it comprises a linear pile of layers that allows you to configure models layer-by-layer for most problems. The sequential model is very simple to use, however, it is limited in its topology. The limitation comes from the fact that you are not able to configure models with shared layers or have multiple inputs or outputs.

Functional API
Alternatively, the Functional API is ideal for creating complex models, that require extended flexibility. It allows you to define models that feature layers connect to more than just the previous and next layers. Models are defined by creating instances of layers and connecting them directly to each other in pairs, Actually, with this model you can connect layers to any other layer. With this model creating complex networks such as siamese networks, residual networks, multi-input/multi-output models, directed acyclic graphs (DAGs), and models with shared layers becomes possible.

Tensorflow:-What is tensorflow "https://youtu.be/Bn2gITckiD0" see this vedio for complete info.
Tensorflow works on the basis of computaional_graph formate i,e,, through (nodes & lines).

*Tensorflow = Tensor + flow

*Tensor:-tensor means multidimension array.i,e,, 1D tensor(only one

rows/columns) ,2D tensor & nD tensor (channel_first or channel_last

*flow:-tensorflow ecicuted in computational graph formate (session code) i,e,, explain in the vedio.

*Tensorflow construction involves 2 steps:-
1st step:-Buiiding a computational_graph:-it is an series of tensorflow operation arranged as nodes in graph.Each node takes one or more tensors as i/p & produce tensors as o/p.
2nd step:-Running a computational_graph:-"see the vedio for more info"

*channel_first and channel_last:-This is nothing but Tensor matrix representation "https://viralfsharp.com/2019/03/25/supercharging-object-detection-in-video-tensorrt-5/"

*Restricted Boltzmann Machines:-"https://towardsdatascience.com/restricted-boltzmann-machines-simplified-eab1e5878976"

*Paceholder , variable & constant:-"https://youtu.be/tudy9xTCXVc" See this vedio for complete information.

*Epsilon:-to prevent denominator become zero.When there's a division operation taking place, it's often added to the denominator to prevent a divide by zero error. Epsilon is small value (1e-07 in TensorFlow Core v2.2.0) that makes very little difference to the value of the denominator, but ensures that it isn't equal to exactly zero.

*Shape(None,2,3):-for column_last:-Means array of shape, (None_Rows means no limit for number of rows i,e,, infinite), (2 means there are 2 columns) & (3 means 3 channels)

*symbolic:-keras.backend.symbolic(func):-It can be defined as a decorator, which is utilized in TensorFlow 2.0 for entering the Keras graph.
Arguments:-func
func: It refers to a function that is used to decorate.
Returns:-o/p
It returns a decorated function.
function is a decorator function provided by Tensorflow 2.0 that converts regular python code to a callable Tensorflow graph function(it allows to perform calling & called function in the computational graph), which is usually more performant and python independent. It is used to create portable Tensorflow models

*eager:-keras.backend.eager(func) :-it ececute without building computational graph.Eager execution is a powerful execution environment that evaluates operations immediately. It does not build graphs, and the operations return actual values instead of computational graphs to run later. With Eager execution, TensorFlow calculates the values of tensors as they occur in your code.

*flotx:-set_floatx
keras.backend.set_floatx(floatx)
It is used to set the default float type value.
Arguments:-
floatx: It refers to a string of float type, such as 'float16', 'float32', or
'float64'.

*cast_to_floatx:-
keras.backend.cast_to_floatx(x)
It is used for casting a Numpy array to the default Keras float type.
Argument
x: It refers to the Numpy array.
Returns
It returns the same Numpy array that is being casted to its new type.
Ex:-
 from keras import backend as K
>>> K.floatx()
'float32'
>>> arr = numpy.array([1.0, 2.0], dtype='float64')
>>> arr.dtype
dtype('float64')
>>> new_arr = K.cast_to_floatx(arr)
>>> new_arr
array([1., 2.], dtype=float32) //// It returns the same Numpy array that is being
casted to its new type.
>>> new_arr.dtype
dtype('float32')

*image_data_format:-By defaultly it will give
keras.backend.image_data_format()
It is used to returns the default image data format convention.
Returns
It returns a string either of 'channels_first' or 'channels_last'
code:-
>>> keras.backend.image_data_format()
'channels_first'

*set_image_data_format:-We can assign data_format as we need.
keras.backend.set_image_data_format(data_format)
This function is used for setting up the data format convention's value.
Arguments
data_format: It can be defined as a string either of 'channels_first' or
'channels_last'.
Example
>>> from keras import backend as K
>>> K.image_data_format()
'channels_first'
>>> K.set_image_data_format('channels_last')

```
>>> K.image_data_format()
'channels_last'
```

For creating Training & Testing Phase:-

learning_phase:-

keras.backend.learning_phase()

It outputs the flag of a learning phase, which refers to a bool tensor (0 = test, 1 = train) to be passed as an input to any of the Keras function that utilizes a distinct behavior both at training and testing time.

Returns

It returns a scalar integer tensor or Python integer of the learning phase.

set_learning_phase:-we need assign either training or testing

keras.backend.set_learning_phase(value)

It is used to set a fixed value to the learning phase.

Arguments

value: It can be defined as an integer that represents the learning phase value to be either 0 or 1.

Raises ##means it shows an error i,e,, we can't select either 0 or 1

ValueError: It is raised if the value is neither 0 nor 1.

Clearing old graphs:-

clear_session:-

keras.backend.clear_session()

It is used for destroying the current graph of Keras and creating a new one. It is very useful as it avoids clutter from old models/layers.

Example1: calling clear_session() while creating models in a loop.

```
# Without `clear_session()`, each iteration of this loop will
# slightly increase the size of the global state managed by Keras  i,e,,
for _ in range(100):
        model = tf.keras.Sequential([tf.keras.layers.Dense(10) for _ in range(10)])
 ##Sequential is a keras model i,e,, layers are connected layer by layer or stockwise

 ##layers.Dense means 10 (Dense or Hidden layers in the nural network)

  # With `clear_session()` called at the beginning,
  # Keras starts with a blank state at each iteration
  # and memory consumption is constant over time. i,e,,
for _ in range(100):
tf.keras.backend.clear_session()
  model = tf.keras.Sequential([tf.keras.layers.Dense(10) for _ in range(10)])
```

Example2: resetting the layer name generation counter.

```
>>> import tensorflow as tf
```

```
>>> layers = [tf.keras.layers.Dense(10) for _ in range(10)]
>>> new_layer = tf.keras.layers.Dense(10)
>>> print(new_layer.name)
dense_10
>>> tf.keras.backend.set_learning_phase(1)
>>> print(tf.keras.backend.learning_phase())
1
>>> tf.keras.backend.clear_session()
>>> new_layer = tf.keras.layers.Dense(10)
>>> print(new_layer.name)
dense
```

*is_sparse:-
keras.backend.is_sparse(tensor)
It is used to return whether a tensor is a sparse tensor.
Arguments
tensor: It refers to an instance of tensor.
Returns
It returns a Boolean.
Example

```
>>> from keras import backend as K
>>> a = K.placeholder((2, 2), sparse=False)
>>> print(K.is_sparse(a))
False
>>> b = K.placeholder((2, 2), sparse=True)
>>> print(K.is_sparse(b))
True
```

*to_dense
keras.backend.to_dense(tensor)
It is used in conversion of a sparse tensor to a dense tensor and returns it.In order to make efficient memory space.bcz spare tensor has almost zero values.it will conpressed to dense tensor.
Arguments
tensor: It refers to an instance of a tensor(potentially sparse).
Returns
It returns a dense tensor.
Example

```
>>> from keras import backend as K
>>> b = K.placeholder((2, 2), sparse=True)
>>> print(K.is_sparse(b))
True
>>> c = K.to_dense(b)
>>> print(K.is_sparse(c))
False
```

Parameters of Tensorflow:-
1)variable

keras.backend.variable(value, dtype=None, name=None, constraint=None)
It helps in instantiating a variable and returning it.
Arguments
value: It can be defined as a numpy array that represents tensor's initial value.
dtype: It refers to the type of a Tensor.
name: For a tensor it indicates a string name.
constraint: It refers to an optional projection function that is implemented on the variable after updating an optimizer.
Returns
It returns an instance of a variable that comprising of a Keras metadata.
Example

```
>>> from keras import backend as K
>>> val = np.array([[1, 2], [3, 4]])
>>> kvar = K.variable(value=val, dtype='float64', name='example_var')
>>> K.dtype(kvar)
'float64'
>>> print(kvar)                 #it won't print the array,bcz in the tensorflow
there is a computational_graph therefore it print only name of tensor_variable
example_var
>>> K.eval(kvar)                # in tensorflow in order to get result we need to
use 1)session or 2)eval(evaluation)
array([[ 1.,  2.],
       [ 3.,  4.]])
```

*is_variable:- to check variable
keras.backend.is_variable(x)

2)constant:-
keras.backend.constant(value, dtype=None, shape=None, name=None)
It lead to the creation of a unique tensor.
Arguments
value: It refers to constant value or a list.
dtype: It refers to the type of a Tensor.
name: For a tensor it indicates a string name.
shape: It can be defined as a dimensionality of the resulting tensor, which is an optional.
Returns
It also return a unique Tensor.

3)KERAS:-Here Keras is composed of I/P , Hidden and O/P these are called symbolic tensor. apart from that nothing will be keras
is_keras_tensor
keras.backend.is_keras_tensor(x)
It outputs whether x is a Keras tensor or not. A "Keras tensor" is a tensor that was returned by a Keras layer, (Layer class) or by Input.
Arguments
x: It refers to a candidate tensor.
Returns
It returns a Boolean that represents whether the argument is a Keras tensor or not.

Raises
It raises a ValueError if x is not a symbolic tensor.
Example
```
>>> from keras import backend as K
>>> from keras.layers import Input, Dense
>>> np_var = numpy.array([1, 2])
>>> K.is_keras_tensor(np_var)    # A numpy array is not a symbolic tensor.
ValueError
>>> k_var = tf.placeholder('float32', shape=(1,1))
>>> K.is_keras_tensor(k_var)   # A variable indirectly created outside of keras is
not a Keras tensor.bcz placeholder is a outside variable.
False
>>> keras_var = K.variable(np_var)
>>> K.is_keras_tensor(keras_var)  # A variable created with the keras backend is not
a Keras tensor.
False
>>> keras_placeholder = K.placeholder(shape=(2, 4, 5))
>>> K.is_keras_tensor(keras_placeholder)  # A placeholder is not a Keras tensor.
False
>>> keras_input = Input([10])
>>> K.is_keras_tensor(keras_input) # An Input is a Keras tensor.
True
>>> keras_layer_output = Dense(10)(keras_input)
>>> K.is_keras_tensor(keras_layer_output)  # Any Keras layer output is a Keras
tensor.
True
```

4)Placeholders:-

Tensors whose values are unknown during the graph construction but passed as input during session

placeholder

keras.backend.placeholder(shape=None, ndim=None, dtype=None, sparse=False, name=None)

It helps in instantiating a placeholder tensor and returning it.

Arguments

shape: It can be defined as a tuple integer, which may incorporate None entries helps in representing the placeholder's Shape.

ndim: It refers to the number of tensor's axes, which specifies at least one of {shape, ndim}. The shape is used, if both are specified.

dtype: It defines the type of Placeholder.

sparse: It can be defined as a Boolean that represents whether or not the placeholder to have a sparse type.

name: It is an optional argument that defines a string for the placeholder's name.

Returns

It returns an instance of a Tensor by including a Keras metadata.

Example

```
>>> from keras import backend as K
>>> input_ph = K.placeholder(shape=(2, 4, 5))
>>> input_ph._keras_shape
```

```
(2, 4, 5)
>>> input_ph
<tf.Tensor 'Placeholder_4:0' shape=(2, 4, 5) dtype=float32>
is_placeholder
keras.backend.is_placeholder(x)
It returns if x is a placeholder or not.
Arguments
x: It can be defined as a candidate placeholder.
Returns
It returns a Boolean.


*shape:-
keras.backend.shape(x)
It outputs the symbolic shape of a tensor or variable.
Arguments
x: It refers to a tensor or variable.
Returns
It returns a tensor of symbolic shape.

*int_shape:-
keras.backend.int_shape(x)
It can be defined as a tuple of int or None entries that outputs the tensor or a
variable's shape.
Arguments
x: It refers to either a tensor or variable.
Returns
It either returns a tuple of integers or None entries.

*ndim:-
keras.backend.ndim(x)
It refers to an integer that are returned as number of axes within a tensor.
Arguments
x: It can be either defined as a tensor or variable.
Returns
It outputs the number of axes as an integer value.
Example
>>> from keras import backend as K
>>> inputs = K.placeholder(shape=(2, 4, 5))   ##shape(2,4,5) it means 2 rows,4
columns & 5 channels i,e,, 3D
>>> val = np.array([[1, 2], [3, 4]])
>>> kvar = K.variable(value=val)
>>> K.ndim(inputs)
3                                       #3D
>>> K.ndim(kvar)
2                                       #2D


5)Evaluation/eval:-
keras.backend.eval(x)
```

It helps in evaluating tensor value.
Arguments
x: It can be defined as a tensor.
Returns
It outputs a Numpy array.
Example
```
>>> from keras import backend as K
>>> kvar = K.variable(np.array([[1, 2], [3, 4]]), dtype='float32')
>>> K.eval(kvar)
array([[ 1.,  2.],
       [ 3.,  4.]], dtype=float32)
```

6)Zeros:-
keras.backend.zeros(shape, dtype=None, name=None)
It helps in instantiation of those variables that are all-zeros followed by returning it.
Arguments
shape: It can be defined as a tuple of integers that represents the returned Keras variable's shape.
dtype: It refers to a string that corresponds to the returned Keras variable's data type.
name: It refers to the string that represent the returned Keras variable's name.
Returns
It returns a variable that includes the Keras metadata, which is filled with 0.0. It should be noted that if it is symbolic n shape, then a variable cannot be returned rather a dynamic-shaped tensor will be returned.
Example
```
>>> from keras import backend as K
>>> kvar = K.zeros((3,4))
>>> K.eval(kvar)
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]], dtype=float32)
```

7)Ones:-
Simillar to Zeros

8)Identity Matrix/ eye(I):
keras.backend.eye(size, dtype=None, name=None)
It helps in the instantiation of an identity matrix followed by returning it.
Arguments
size: It can be defined either as a tuple defining the number of rows and columns or an integer that represents the number of rows.
dtype: It refers to a string that corresponds to the returned Keras variable's data type.
name: It refers to the string that represent the returned Keras variable's name.
Returns
It outputs a Keras variable that represents an identity matrix.
Example
```
>>> from keras import backend as K
```

```
>>> K.eval(K.eye(3))
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]], dtype=float32)
>>> K.eval(K.eye((2, 3)))
array([[1., 0., 0.],
       [0., 1., 0.]], dtype=float32)
```

9)zeros_like:-
keras.backend.zeros_like(x, dtype=None, name=None) means all zeros values.
It returns a variable of Keras filled with all zeros that constitutes a shape of x.
Example

```
>>> from keras import backend as K
>>> kvar = K.variable(np.random.random((2,3)))
>>> kvar_zeros = K.zeros_like(kvar)
>>> K.eval(kvar_zeros)
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]], dtype=float32)
```

10)ones_like:-
Similar to zers_like but all values are 1

11)Product/dot:-
keras.backend.dot(x, y)
It returns a tensor by either multiplying 2 tensors or variable.
While multiplying an nD tensor to another nD tensor, a Theano behavior is reproduced. (e.g. (2, 3) * (4, 3, 5) -> (2, 4, 5))
Arguments
x: It refers to a tensor or variable.
y: It refers to a tensor or variable.
Returns
It returns a tensor, which is produced after undergoing a dot product between x and y.
Examples

```
# dot product between tensors
>>> x = K.placeholder(shape=(2, 3))
>>> y = K.placeholder(shape=(3, 4))
>>> xy = K.dot(x, y)                            ##x*y
>>> xy
<tf.Tensor 'MatMul_9:0' shape=(2, 4) dtype=float32>
# dot product between tensors
>>> x = K.placeholder(shape=(32, 28, 3))
>>> y = K.placeholder(shape=(3, 4))
>>> xy = K.dot(x, y)
>>> xy
<tf.Tensor 'MatMul_9:0' shape=(32, 28, 4) dtype=float32>
# Theano-like behavior example
>>> x = K.random_uniform_variable(shape=(2, 3), low=0, high=1)
>>> y = K.ones((4, 3, 5))
>>> xy = K.dot(x, y)
```

```
>>> K.int_shape(xy)
(2, 4, 5)
```

12)Keras Sequential Model:-

```
model=Sequential ()
model.add(Dense(32, input_shape=(784,)))        ##It means that 32 neurons with 784
rows and 0 columns(i,e,, List) input parameters in the first hidden layers (by
                                    consequently it will take another  hidden layers)
model=Sequential ()
model.add(Dense(32, input_dim=784))             ####It means that 32 neurons with
784 rows and 0 columns(i,e,, List) input parameters in the first hidden layers (by
                                                      consequently it will take
another  hidden layers)
model = Sequential()
model.add(Dense(32, batch_input_shape=(None, 784)))     # note that batch dimension
is "None" here,# so the model will be able to process batches of any size.
```

13)Optimization:-

RMSProp, root mean square propagation, is an optimization algorithm/method designed for Artificial Neural Network (ANN) training.

14)(/=)Division Assignment:-

Description

Divides the variable by a value and assigns the result to that variable.

```
>>> a = 10
>>> a /= 5
>>> a
2
```

Keras Layers:-

Keras encompasses a wide range of predefined layers as well as it permits you to create your own layer. It acts as a major building block while building a Keras model. In Keras, whenever each layer receives an input, it performs some computations that result in transformed information. The output of one layer is fed as input to the other layer.

*Keras Core layer comprises of:-

1)dense layer:-output = activation(dot(input, kernel) + bias)
which is a dot product plus bias.(In any neural network, a dense layer is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to every neuron of its preceding layer. This layer is the most commonly used layer in artificial neural network networks.)

(Keras Dense Layer Operation

The dense layer function of Keras implements following operation –

```
output = activation(dot(input, kernel) + bias)          ##it is a dot product plus
bias
```

In the above equation, activation is used for performing element-wise activation and the kernel is the weights matrix created by the layer, and bias is a bias vector created by the layer.

Keras dense layer on the output layer performs dot product of input tensor and weight kernel matrix.)

2)activation:- layer that transfers a function or neuron shape.(In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs.)

3)dropout layer:- which randomly at each training update, sets a fraction of input unit to zero so as to avoid the issue of overfitting.

4)lambda layer:-that wraps an arbitrary expression just like an object of a Layer. Lambda layer exists so that arbitrary expressions can be used as a Layer when constructing Sequential and Functional API models.

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

6)downscaling layer:- which is mainly known as pooling, runs from 1D to 3D. It also includes the most common variants, such as max and average pooling.
The main purpose of pooling is to reduce the size of feature maps, which in turn makes computation faster because the number of training parameters is reduced. The pooling operation summarizes the features present in a region, the size of which is determined by the pooling filter.
*There are two types of Pooling: Max Pooling and Average Pooling . Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel
*kernel:- represent the weight data.
*dot represent numpy dot product of all input and its corresponding weights.
*bias represent a biased value used in machine learning to optimize the model.
*activation represent the activation function.

7)The noise layer :-eradicates the issue of overfitting. The recurrent layer that includes simple, gated, LSTM, etc. are implemented in applications like language processing.

Following are the number of common methods that each Keras layer have:-

get_weights(): It yields the layer's weights as a numpy arrays list.
set_weights(weights): It sets the layer's weight with the similar shapes as that of the output of get_weights() from numpy arrays list.
get_config(): It returns a dictionary that includes the layer's configuration, so as to instantiate from its config through;

code:-
```
layer = Dense(32)
```

```
config = layer.get_config()
reconstructed_layer = Dense.from_config(config)
```

In case when layer isn't the shared layer or we can say the layer comprises of individual node, then we can get its input tensor, output tensor, input shape and output shape through the followings; shred layer is a layer we can use that layer by calling it with new layer.

input
output
input_shape
output_shape

Else, if the layer encompasses several nodes then in that case you can use the following methods given below;

get_input_at(node_index)
get_output_at(node_index)
get_input_shape_at(node_index)
get_output_shape_at(node_index)