Keras Architecture

Tushar B. Kute,
http://tusharkute.com

Keras Architecture

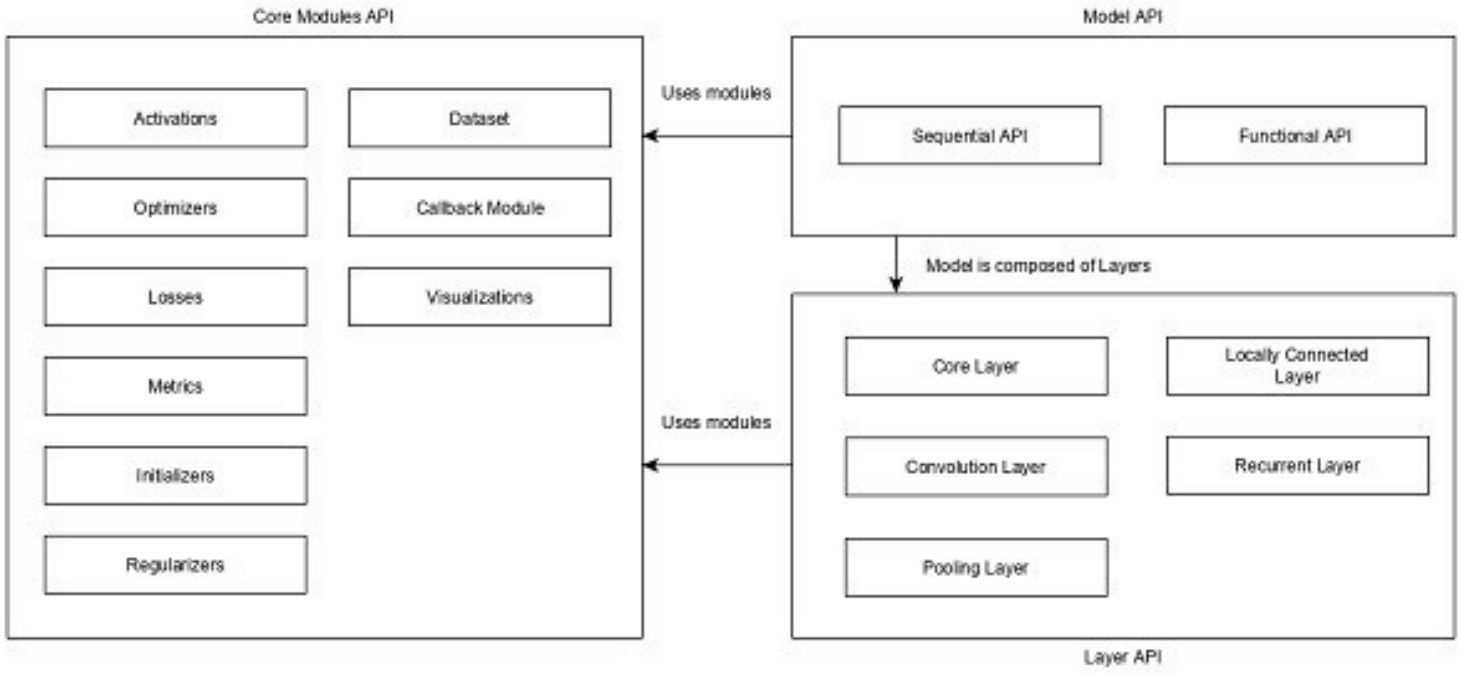
- Keras provides a complete framework to create any type of neural networks. Keras is innovative as well as very easy to learn.
- It supports simple neural network to very large and complex neural network model.
- Keras API can be divided into three main categories –
  - Model
  - Layer
  - Core Modules

Keras Architecture

- In Keras, every ANN is represented by Keras Models. In turn, every Keras Model is composition of Keras Layers and represents ANN layers like input, hidden layer, output layers, convolution layer, pooling layer, etc.,
- Keras model and layer access Keras modules for activation function, loss function, regularization function, etc.,
- Using Keras model, Keras Layer, and Keras modules, any ANN algorithm (CNN, RNN, etc.,) can be represented in a simple and efficient manner.

Keras Architecture

- The following diagram depicts the relationship between model, layer and core modules –



Model

- Keras Models are of two types as mentioned below –
- Sequential Model – Sequential model is basically a linear composition of Keras Layers. Sequential model is easy, minimal as well as has the ability to represent nearly all available neural networks.
- A simple sequential model is as follows –

```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential( )
model.add(Dense(512, activation = 'relu', input_shape = (784,)))
```

Model

- Line 1 imports Sequential model from Keras models
- Line 2 imports Dense layer and Activation module
- Line 4 create a new sequential model using Sequential API
- Line 5 adds a dense layer (Dense API) with relu activation (using Activation module) function.

Model

- Sequential model exposes Model class to create customized models as well. We can use sub-classing concept to create our own complex model.
- Functional API – Functional API is basically used to create complex models.

Layer

- Each Keras layer in the Keras model represent the corresponding layer (input layer, hidden layer and output layer) in the actual proposed neural network model.
- Keras provides a lot of pre-build layers so that any complex neural network can be easily created. Some of the important Keras layers are specified below,
  - Core Layers
  - Convolution Layers
  - Pooling Layers
  - Recurrent Layers

Layer

- A simple python code to represent a neural network model using sequential model is as follows –

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation = 'softmax'))
```

Layer

- Line 1 imports Sequential model from Keras models
- Line 2 imports Dense layer and Activation module
- Line 4 create a new sequential model using Sequential API
- Line 5 adds a dense layer (Dense API) with relu activation (using Activation module) function.
- Line 6 adds a dropout layer (Dropout API) to handle over-fitting.
- Line 7 adds another dense layer (Dense API) with relu activation (using Activation module) function.
- Line 8 adds another dropout layer (Dropout API) to handle over-fitting.
- Line 9 adds final dense layer (Dense API) with softmax activation (using Activation module) function.

Layer

- Keras also provides options to create our own customized layers.
- Customized layer can be created by sub-classing the Keras.Layer class and it is similar to sub-classing Keras models.

Core Modules

- Keras also provides a lot of built-in neural network related functions to properly create the Keras model and Keras layers. Some of the function are as follows –
- Activations module – Activation function is an important concept in ANN and activation modules provides many activation function like softmax, relu, etc.,
- Loss module – Loss module provides loss functions like mean_squared_error, mean_absolute_error, poisson, etc.,
- Optimizer module – Optimizer module provides optimizer function like adam, sgd, etc.,
- Regularizers – Regularizer module provides functions like L1 regularizer, L2 regularizer, etc.,

Available Modules

- Initializers – Provides a list of initializers function. We can learn it in details in Keras layer chapter. during model creation phase of machine learning.
- Regularizers – Provides a list of regularizers function. We can learn it in details in Keras Layers chapter.
- Constraints – Provides a list of constraints function. We can learn it in details in Keras Layers chapter.

Available Modules

- Activations – Provides a list of activator function.
- Losses – Provides a list of loss function.
- Metrics – Provides a list of metrics function.
- Optimizers – Provides a list of optimizer function.

Available Modules

- Callback – Provides a list of callback function. We can use it during the training process to print the intermediate data as well as to stop the training itself (EarlyStopping method) based on some condition.
- Text processing – Provides functions to convert text into NumPy array suitable for machine learning. We can use it in data preparation phase of machine learning.
- Image processing – Provides functions to convert images into NumPy array suitable for machine learning. We can use it in data preparation phase of machine learning.

Available Modules

- Sequence processing – Provides functions to generate time based data from the given input data. We can use it in data preparation phase of machine learning.
- Backend – Provides function of the backend library like TensorFlow and Theano.
- Utilities – Provides lot of utility function useful in deep learning.

Modules - utils

- to_categorical
  - It is used to convert class vector into binary class matrix.

```
>>> from keras.utils import to_categorical
>>> labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> to_categorical(labels)
```

Layers

- A Keras layer requires shape of the input (input_shape) to understand the structure of the input data, initializer to set the weight for each input and finally activators to transform the output to make it non-linear.
- In between, constraints restricts and specify the range in which the weight of input data to be generated and regularizer will try to optimize the layer (and the model) by dynamically applying the penalties on the weights during optimization process.

Layers

- To summarise, Keras layer requires below minimum details to create a complete layer.
  - Shape of the input data
  - Number of neurons / units in the layer
  - Initializers
  - Regularizers
  - Constraints
  - Activations

Example:

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers
from keras import regularizers
from keras import constraints
model = Sequential()
model.add(Dense(32, input_shape=(16,), kernel_initializer = 'he_uniform', kernel_regularizer = None, kernel_constraint = 'MaxNorm', activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(8))
```

Dense Layer

- Line 9 creates a new Dense layer and add it into the model.
- Dense is an entry level layer provided by Keras, which accepts the number of neurons or units (32) as its required parameter.
- If the layer is first layer, then we need to provide Input Shape, (16,) as well. Otherwise, the output of the previous layer will be used as input of the next layer.
- All other parameters are optional.

Dense Layer

- First parameter represents the number of units (neurons). input_shape represent the shape of input data.
- kernel_initializer represent initializer to be used. he_uniform function is set as value.
- kernel_regularizer represent regularizer to be used. None is set as value.
- kernel_constraint represent constraint to be used. MaxNorm function is set as value.
- activation represent activation to be used. relu function is set as value.

Basic Concept - Input shape

- In machine learning, all type of input data like text, images or videos will be first converted into array of numbers and then feed into the algorithm.
- Input numbers may be single dimensional array, two dimensional array (matrix) or multi-dimensional array.
- We can specify the dimensional information using shape, a tuple of integers.
- For example, (4,2) represent matrix with four rows and two columns.

Basic Concept – Initializers

- In Machine Learning, weight will be assigned to all input data. Initializers module provides different functions to set these initial weight. Some of the Keras Initializer function are as follows –
- Generates 0 for all input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers
my_init = initializers.Zeros()
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,),
  kernel_initializer = my_init))
```

Basic Concept – Regularizers

- In machine learning, regularizers are used in the optimization phase.
- It applies some penalties on the layer parameter during optimization.
- Keras regularization module provides below functions to set penalties on the layer. Regularization applies per-layer basis only.

Basic Concept – Regularizers

- L1 Regularizer

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import regularizers
my_regularizer = regularizers.l1(0.)
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,), kernel_regularizer = my_regularizer))
```

Basic Concept – Regularizers

- L2 Regularizer

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import regularizers
my_regularizer = regularizers.l2(0.)
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,), kernel_regularizer = my_regularizer))
```

Basic Concept – Activations

- In machine learning, activation function is a special function used to find whether a specific neuron is activated or not.
- Basically, the activation function does a nonlinear transformation of the input data and thus enable the neurons to learn better. Output of a neuron depends on the activation function.
- As you recall the concept of single perception, the output of a perceptron (neuron) is simply the result of the activation function, which accepts the summation of all input multiplied with its corresponding weight plus overall bias, if any available.

Basic Concept – Activations

- Types:
  - linear
  - relu
  - softmax
  - softplus
  - tanh
  - sigmoid

Types of Layers

- Dense Layer
  - Dense layer is the regular deeply connected neural network layer.
- Dropout Layers
  - Dropout is one of the important concept in the machine learning.
- Flatten Layers
  - Flatten is used to flatten the input.
- Reshape Layers
  - Reshape is used to change the shape of the input.

Types of Layers

- Permute Layers
  - Permute is also used to change the shape of the input using pattern.
- RepeatVector Layers
  - RepeatVector is used to repeat the input for set number, n of times.
- Lambda Layers
  - Lambda is used to transform the input data using an expression or function.
- Convolution Layers
  - Keras contains a lot of layers for creating Convolution based ANN, popularly called as Convolution Neural Network (CNN).

Types of Layers

- Pooling Layer
  - It is used to perform max pooling operations on temporal data.
- Locally connected layer
  - Locally connected layers are similar to Conv1D layer but the difference is Conv1D layer weights are shared but here weights are unshared.
- Merge Layer
  - It is used to merge a list of inputs.
- Embedding Layer
  - It performs embedding operations in input layer.

Dense Layer

- Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.
- output = activation(dot(input, kernel) + bias)
- where,
  - input represent the input data
  - 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.

Dense Layer

- Let us consider sample input and weights as below and try to find the result –

  input as 2 x 2 matrix [[1, 2], [3, 4]]

  kernel as 2 x 2 matrix [[0.5, 0.75], [0.25, 0.5]]

  bias value as 0

  activation as linear.

- As we learned earlier, linear activation does nothing.

Dense Layer

```
>>> import numpy as np


>>> input = [ [1, 2], [3, 4] ]
>>> kernel = [ [0.5, 0.75], [0.25, 0.5] ]
>>> result = np.dot(input, kernel)
>>> result array([[1. , 1.75], [2.5 , 4.25]])
```

Dropout Layer

- Dropout is one of the important concept in the machine learning.
- It is used to fix the over-fitting issue. Input data may have some of the unwanted data, usually called as Noise.
- Dropout will try to remove the noise data and thus prevent the model from over-fitting.

Dropout Layer

- Dropout has three arguments and they are as follows –

  keras.layers.Dropout(rate, noise_shape = None, seed = None)

  - rate – represent the fraction of the input unit to be dropped. It will be from 0 to 1.
  - noise_shape represent the dimension of the shape in which the dropout to be applied. For example, the input shape is (batch_size, timesteps, features).
  - Then, to apply dropout in the timesteps, (batch_size, 1, features) need to be specified as noise_shape
  - seed – random seed.

Sequential Model

- The core idea of Sequential API is simply arranging the Keras layers in a sequential order and so, it is called Sequential API.
- Most of the ANN also has layers in sequential order and the data flows from one layer to another layer in the given order until the data finally reaches the output layer.
- A ANN model can be created by simply calling Sequential() API as specified below –

```
from keras.models import Sequential
model = Sequential( )
```

Add Layers

- To add a layer, simply create a layer using Keras layer API and then pass the layer through add() function as specified below –

```
from keras.models import Sequential

model = Sequential()

input_layer = Dense(32, input_shape=(8,))
model.add(input_layer)

hidden_layer = Dense(64, activation='relu');
model.add(hidden_layer)

output_layer = Dense(8)

model.add(output_layer)
```

Access the model

- Keras provides few methods to get the model information like layers, input data and output data. They are as follows –
  - model.layers – Returns all the layers of the model as list.

```
>>> layers = model.layers
>>> layers
```

Serialize the model

- Keras provides methods to serialize the model into object as well as json and load it again later.

  to_json() – Returns the model as an json object.

```
>>> json_string = model.to_json()
>>> json_string
```

Serialize the model

- model_from_json() – Accepts json representation of the model and create a new model.

```
from keras.models import model_from_json
new_model = model_from_json(json_string)
```

Summarise the model

- Understanding the model is very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.
- A summary of the model created in the previous section is as follows –

```
>>> model.summary()
```

Train and Predict the model

- Model provides function for training, evaluation and prediction process. They are as follows –
  - compile – Configure the learning process of the model
  - fit – Train the model using the training data
  - evaluate – Evaluate the model using the test data
  - predict – Predict the results for new input.

Thank you

This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

MITUSkillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com