Employing Layers in Keras Models

Jerry Kurata
CONSULTANT

@jerrykur

Common Methods

get_weights(), set_weights(weights)
get_config(), from_config(config)

input, output
input_shape, output_shape

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

Layers

Expanding number of layers

Over 70 layers

Build most Neural Networks

Keras Layer Groups

Common

Shaping

Merging

Extension

Convolutional – separate module

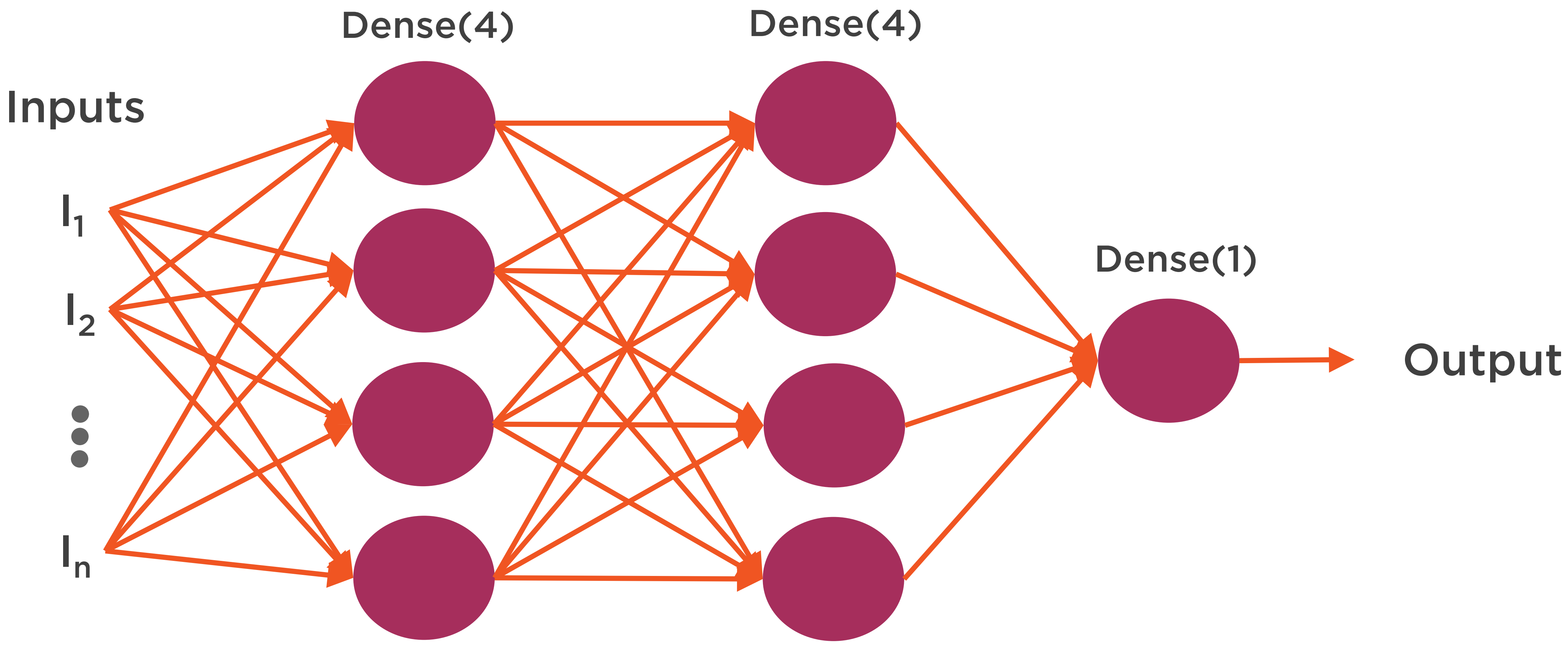
Recurrent – separate module

Common Layers

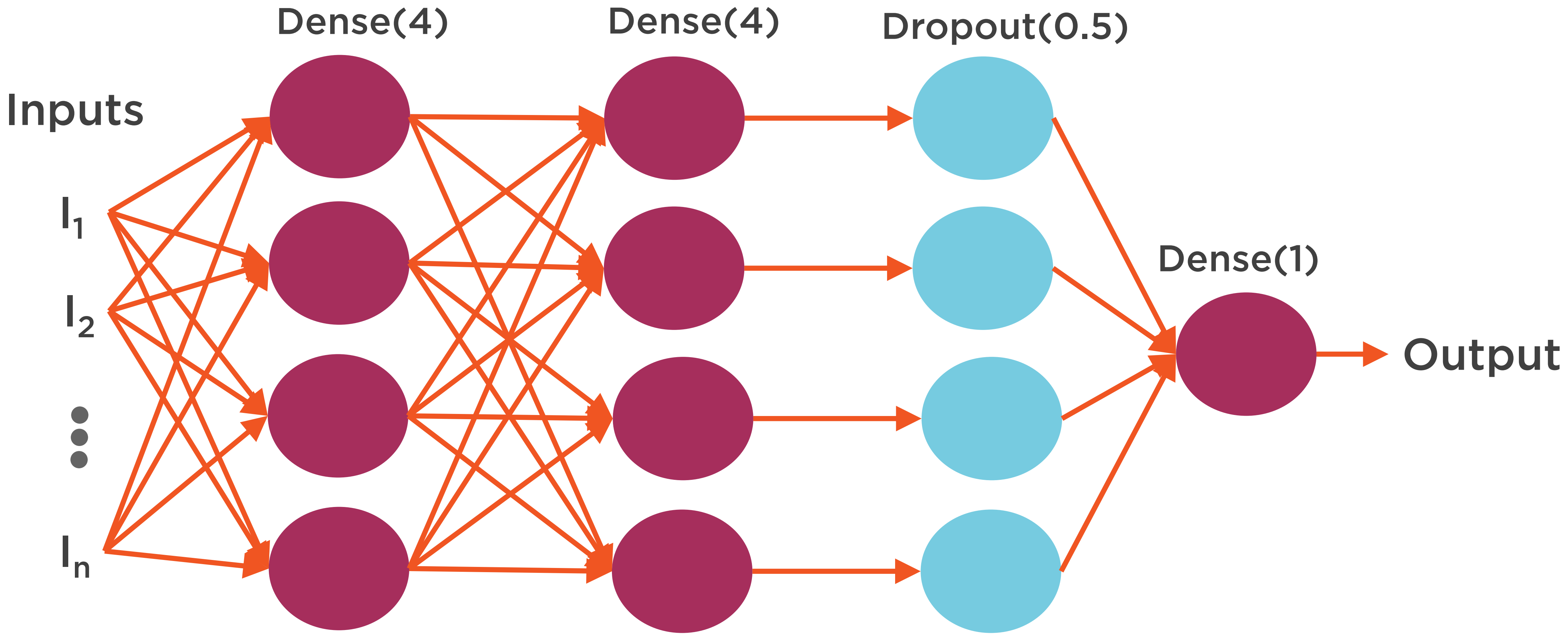
Dense

Dropout

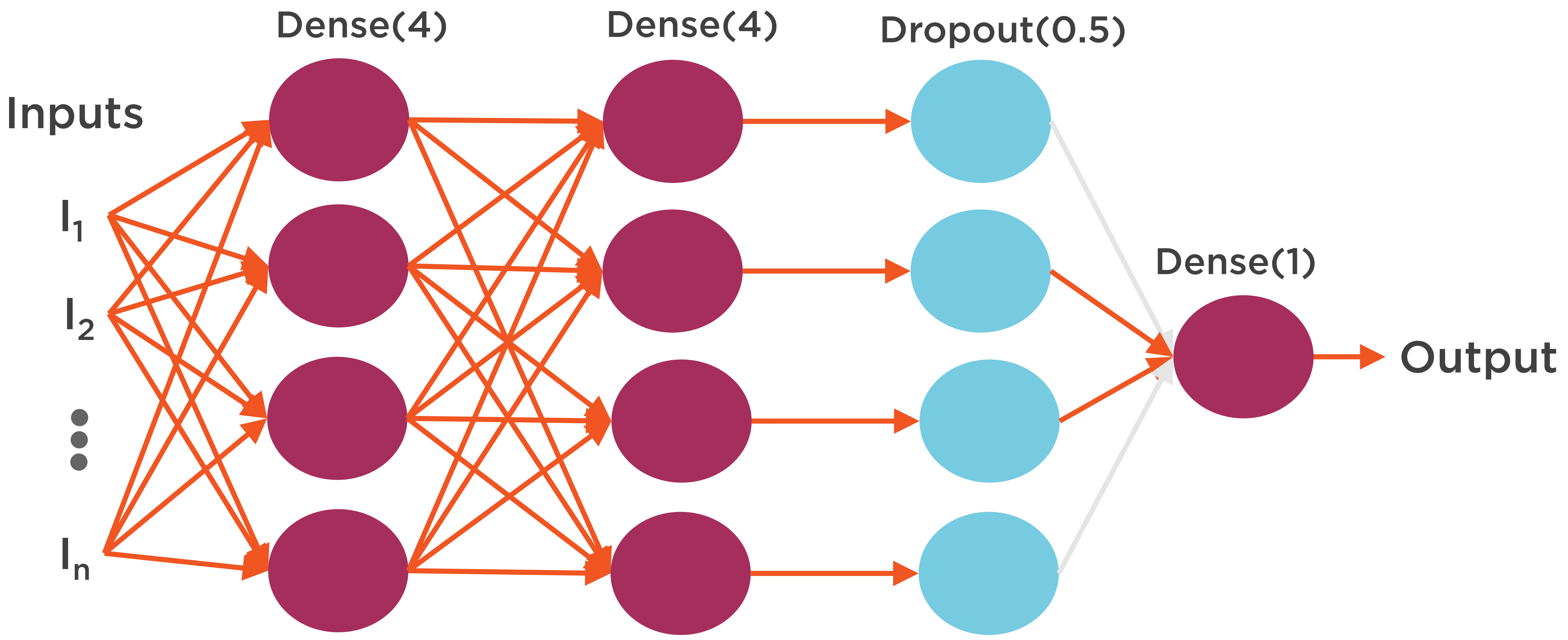
Dense Layer



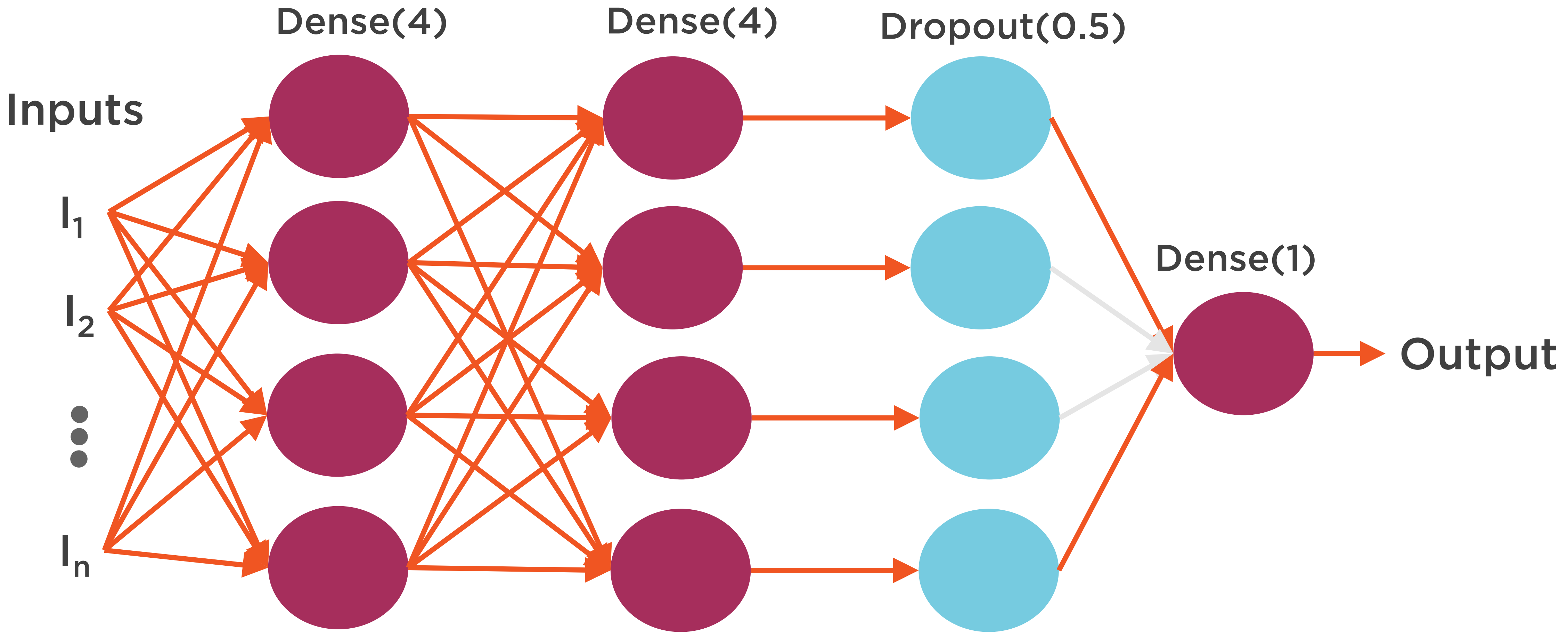
Dropout Layers



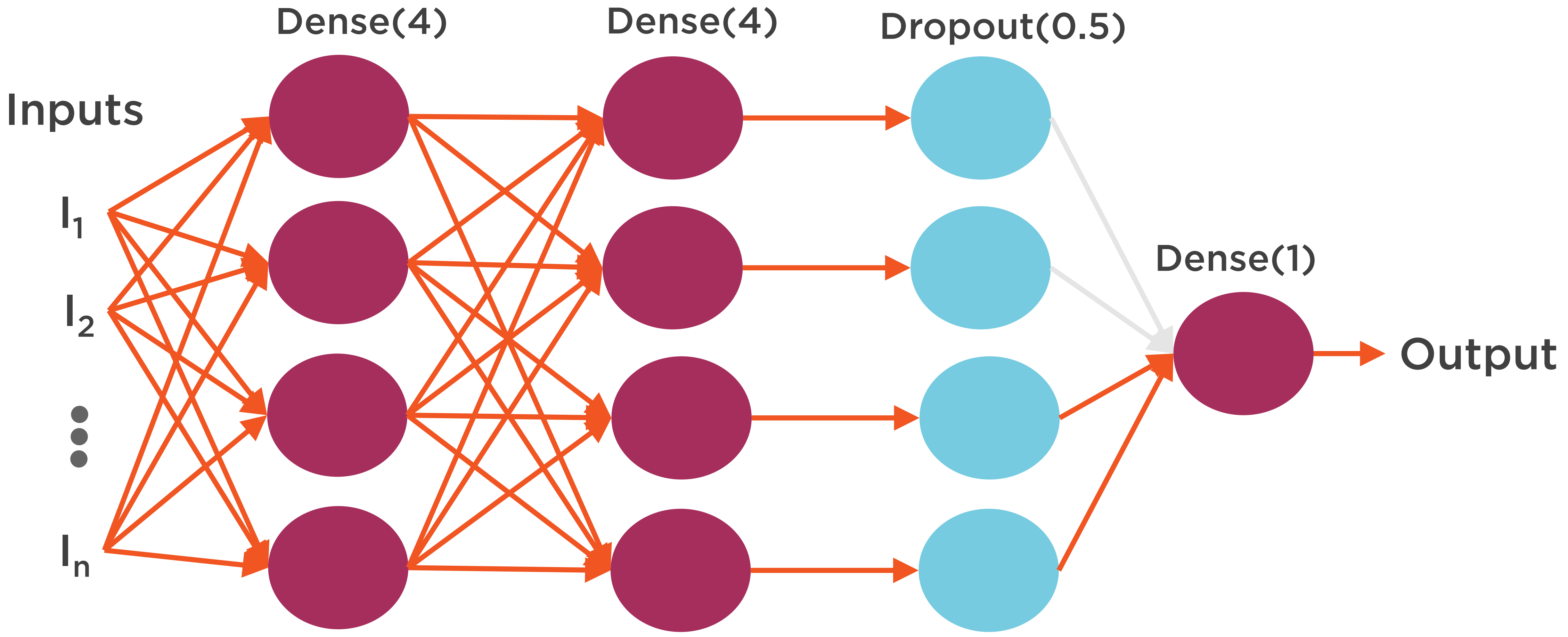
Dropout Layer



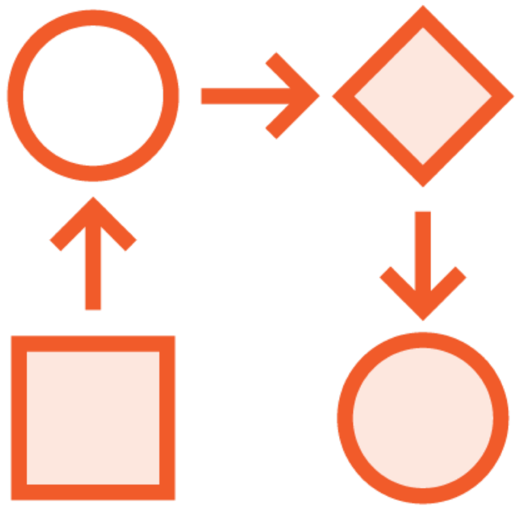
Dropout Layer



Dropout Layers



Shaping Layers



Reshape(target_shape)

- Reshape((2, 3), inputs_shape=(6,))
- input: (None, 6) -> (None, 2, 3)

Flatten()

- Flatten()
- input: (None, 64, 32, 32) -> (None, 65536)

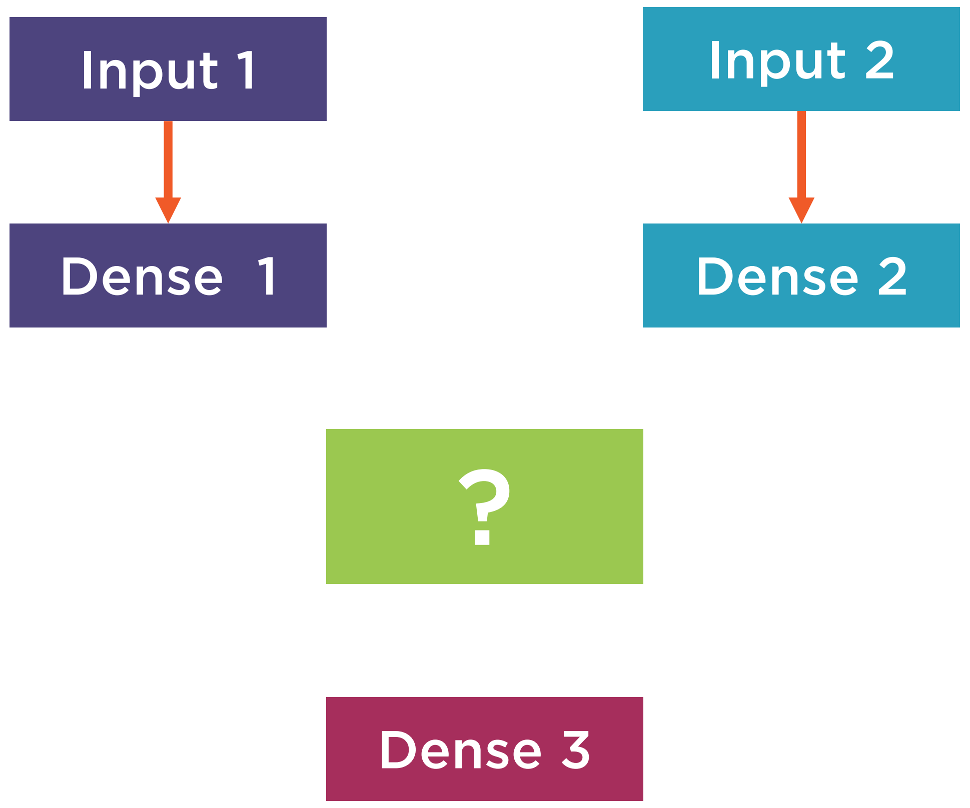
Permute(dims)

- Permute((2,1), input_shape(20,40))
- output -> (None, 40, 20)

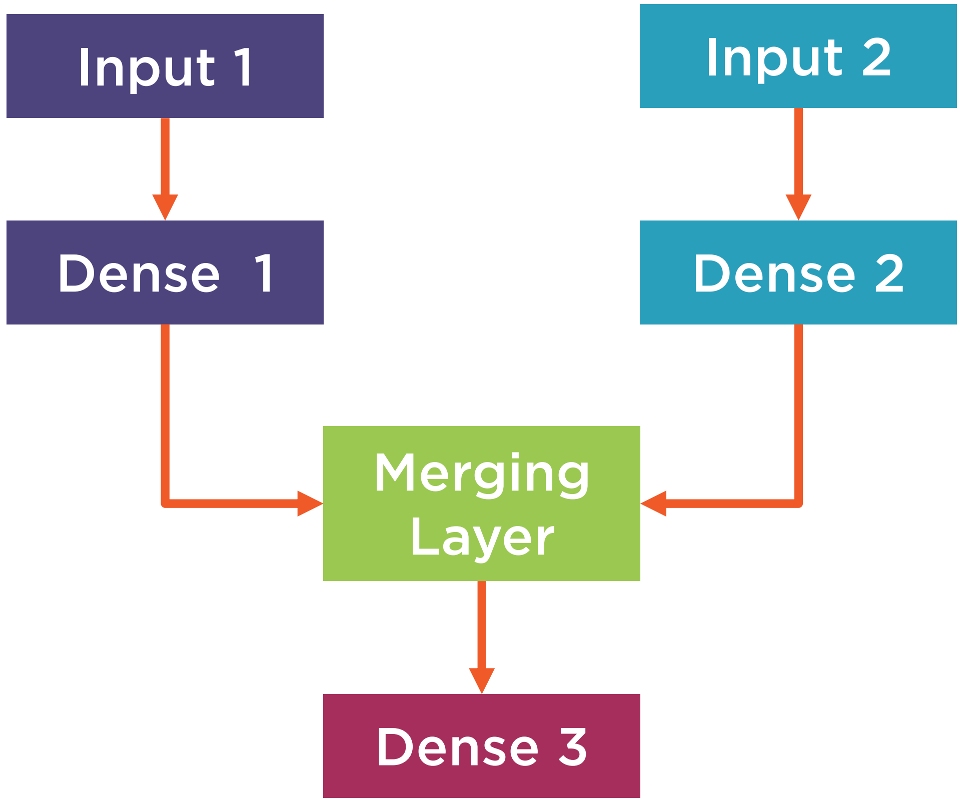
RepeatVector(n)

- RepeatVector(3)
- input (None, 32) -> (None, 3, 32)

Merging Layers



Merging Layers



Merging Layer Features

Different type of “merges”

Take tensors as input

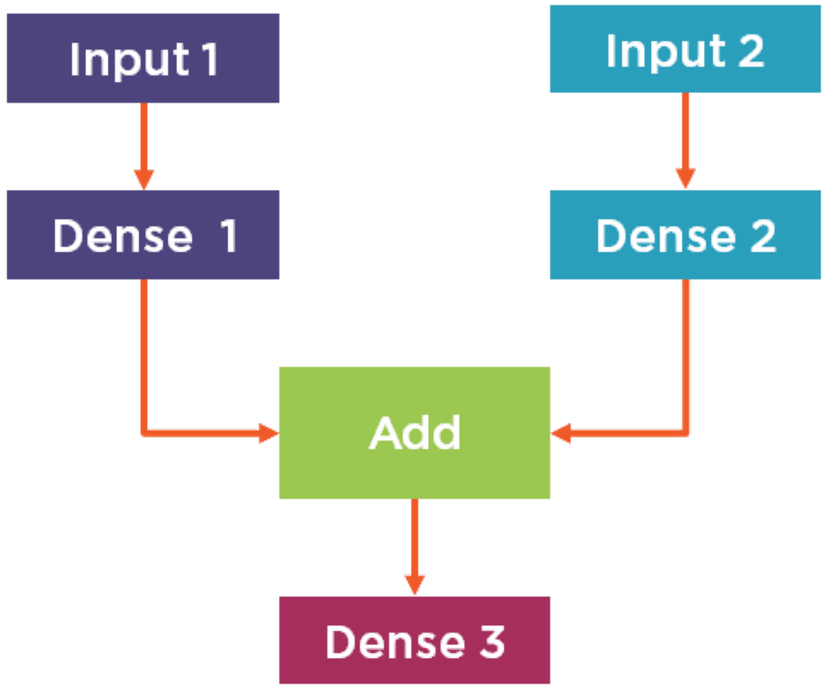
Result is “merged” tensor

Merging Layers

Add	add
Subtract	subtract
Multiply	multiply
Average	average
Maximum	maximum
Concatenate	concatenate
Dot	dot

Subtract()[x1, x2] *is equivalent to* subtract(x1,x2)

Merging Example



```
input1 = keras.layers.Input(shape=(16,), name=‘input_1’)
x1 = keras.layers.Dense(8, activation=‘relu’, name=‘dense_1’)(input1)

input2 = keras.layers.Input(shape=(16,), name=‘input_2’)
x2 = keras.layers.Dense(8, activation=‘relu’, name=‘dense_2’)(input2)
```

```
added = keras.layers.Add()[x1, x2]
# added = keras.layers.add(x1, x2)
```

```
out = keras.layers.Dense(4, name=‘dense_3’)(added)

model = keras.models.Model(inputs=[input1, input2], outputs=out)
```

Extension Layers

Extend functionality

Perform custom tasks

Encapsulate our logic

2 ways

- Lambda Layer
- Custom Layer

Lambda Function

Simple tasks

No trainable weights

Inline or function implementation

Lambda Layer

```
#inline lambda
model.add(Lambda(lambda x: x
                  ** 2))
```

Add squaring layer

```
# function lambda
def sqr(x):
   return x ** 2
```

Define function

```
def sqr_shape(input_shape):
   return input_shape
```

Define shape of output
(Theano only)

```
model.add(Lambda(sqr,
   output_shape =
          sqr_shape))
```

Add layer

Custom Layer

Complex tasks

Trainable weights

Reusable

Must implement methods

Custom Layer - Definition

```
Class MyLayer(Layer):

  def _init__(self,output_dim,**kwargs):
    self.output_dim = output_dim
    super(Mylayer,
         self).__init__(**kwargs)

  def build(self, input_shape):
    self.kernel = self.add_weight(
     name="kernel", shape =
       (input_shape[1],self.output_dim),
     initializer="uniform",
     trainable=True)
    super(MyLayer,self).build(
                       input_shape)

  def call(self,x):
    return K.dot(x, self_kernel)

  def compute_output_shape(self,
      input_shape):
     return(input_shape[0],
          self.output_dim)
```

◀ Initialize class

◀ Define weights to be trained

◀ Specify the layer logic

◀ Define how output shape is determined

Custom Layer - Usage

```
Model.add(MyLayer(…))
```

```
x1 = Dense(64)(in)
x2 = MyLayer(…)(x1)
```

- Usage – Sequential Model
- Usage – Functional API

Summary

Methods common to all layers

Common layers used a lot

Shaping layers shape data

Merging layers merge tensors

Extension layers are user written

Convolutional NN next