Writing your own Keras layers

For simple, stateless custom operations, you are probably better off using `layers.core.Lambda` layers. But for any custom operation that has trainable weights, you should implement your own layer.

Here is the skeleton of a Keras layer. There are only three methods you need to implement:

- `build(input_shape)`: this is where you will define your weights. Trainable weights should be added to the list `self.trainable_weights`. Other attributes of note are: `self.non_trainable_weights` (list) and `self.updates` (list of update tuples (tensor, new_tensor)). For an example of how to use `non_trainable_weights` and `updates`, see the code for the `BatchNormalization` layer.
- `call(x)`: this is where the layer's logic lives. Unless you want your layer to support masking, you only have to care about the first argument passed to `call`: the input tensor.
- `get_output_shape_for(input_shape)`: in case your layer modifies the shape of its input, you should specify here the shape transformation logic. This allows Keras to do automatic shape inference.

```python
from keras import backend as K
from keras.engine.topology import Layer
import numpy as np

class MyLayer(Layer):
    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(MyLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        input_dim = input_shape[1]
        initial_weight_value = np.random.random((input_dim, output_dim))
        self.W = K.variable(initial_weight_value)
        self.trainable_weights = [self.W]

    def call(self, x, mask=None):
        return K.dot(x, self.W)

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

The existing Keras layers provide ample examples of how to implement almost anything. Never hesitate to read the source code!