Keras requires the input shape to be specified in the first layer, but it will automatically infer the shape of all other layers. This means you only have to explicitly set the input dimensions for the first layer.

The first (hidden) layer from above, `model.add(Dense(32, input_dim=X.shape[1]))`, creates 32 nodes which each expect to receive 2-element vectors as inputs. Each layer takes the outputs from the previous layer as inputs and pipes through to the next layer. This chain of passing output to the next layer continues until the last layer, which is the output of the model. We can see that the output has dimension 1.

The activation "layers" in Keras are equivalent to specifying an activation function in the Dense layers (e.g., `model.add(Dense(128)); model.add(Activation('softmax'))` is computationally equivalent to `model.add(Dense(128, activation="softmax")))`), but it is common to explicitly separate the activation layers because it allows direct access to the outputs of each layer before the activation is applied (which is useful in some model architectures).

Once we have our model built, we need to compile it before it can be run. Compiling the Keras model calls the backend (tensorflow, theano, etc.) and binds the optimizer, loss function, and other parameters required before the model can be run on any input data. We'll specify the loss function to be `categorical_crossentropy` which can be used when there are only two classes, and specify `adam` as the optimizer (which is a reasonable default when speed is a priority). And finally, we can specify what metrics we want to evaluate the model with. Here we'll use accuracy.

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

We can see the resulting model architecture with the following command:

```
model.summary()
```