Keras APIs

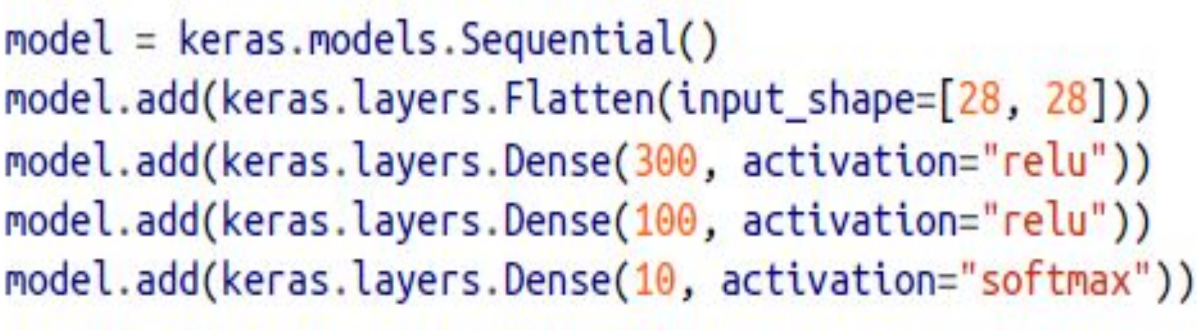
Instructor: Revendranath T

Keras APIs

1. Keras API enable developing simple and complex neural networks architectures
2. Three types of Keras APIs are used to build ANN models
   a. Sequential API
   b. Functional API
   c. Subclassing API

Model using Sequential API

- The most approachable API—it’s basically a Python list.
- Limited to simple stacks of layers
- The Sequential model constitutes
  - Dense layers, and built-in APIs for training, evaluation, and inference—compile(), fit(), evaluate(), and predict().
  - Layer class to create custom layers,
  - Verify the gradient descent using the Tensor-Flow GradientTape in the training loop.
- Sequential APIs approach are not suited to build complex neural network architectures



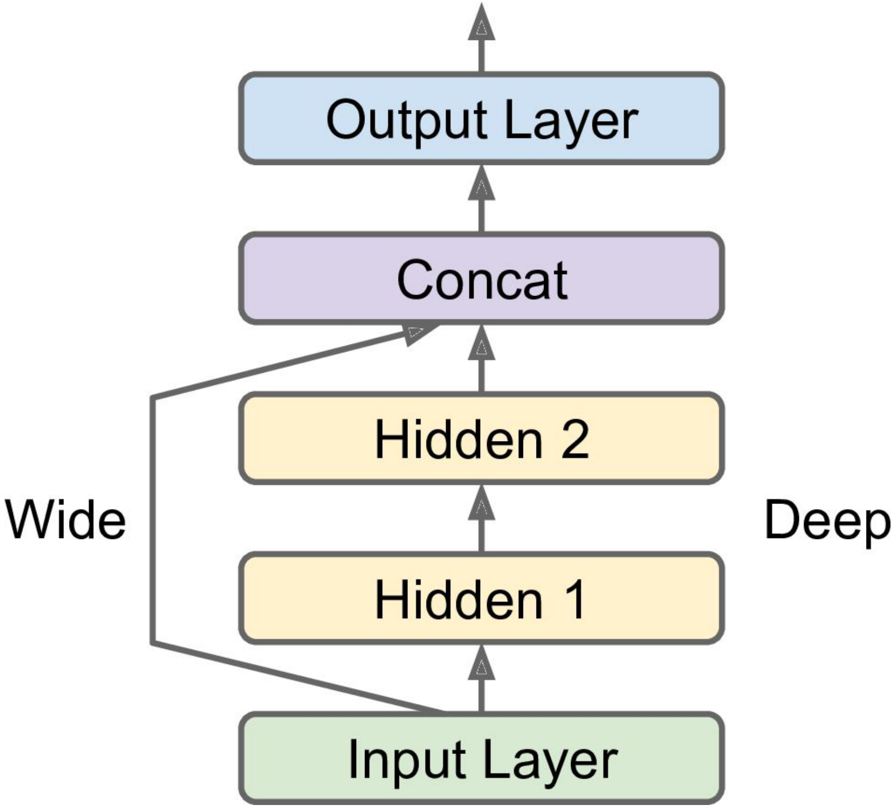
```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

Model using Functional API

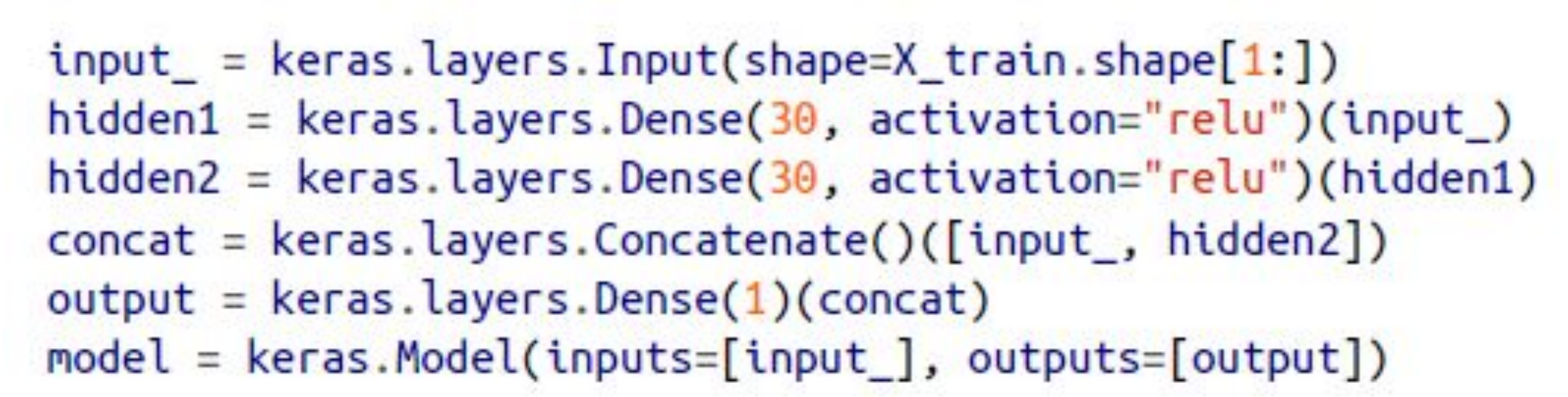
- Focuses on graph-like model architectures.
- Represents a nice mid-point between usability and flexibility
- The most commonly used model-building API.
- Example of non-sequential NN: Wide & Deep neural network [learn both

  deep patterns (using the deep path) and simple rules (through the short path)]

Model using Functional API: Illustration 1

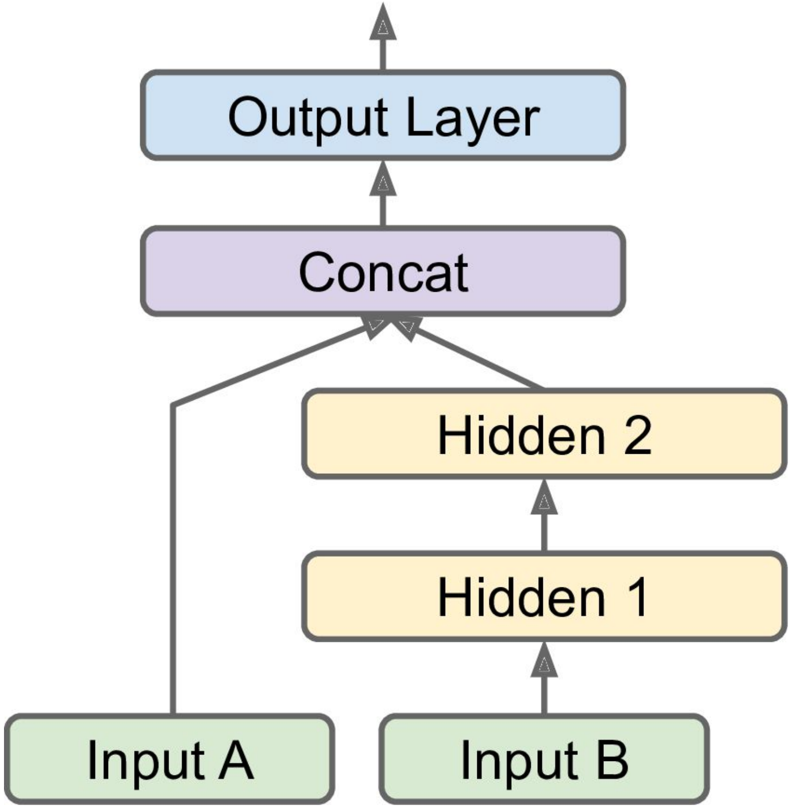


Model using Functional API : Illustration 1

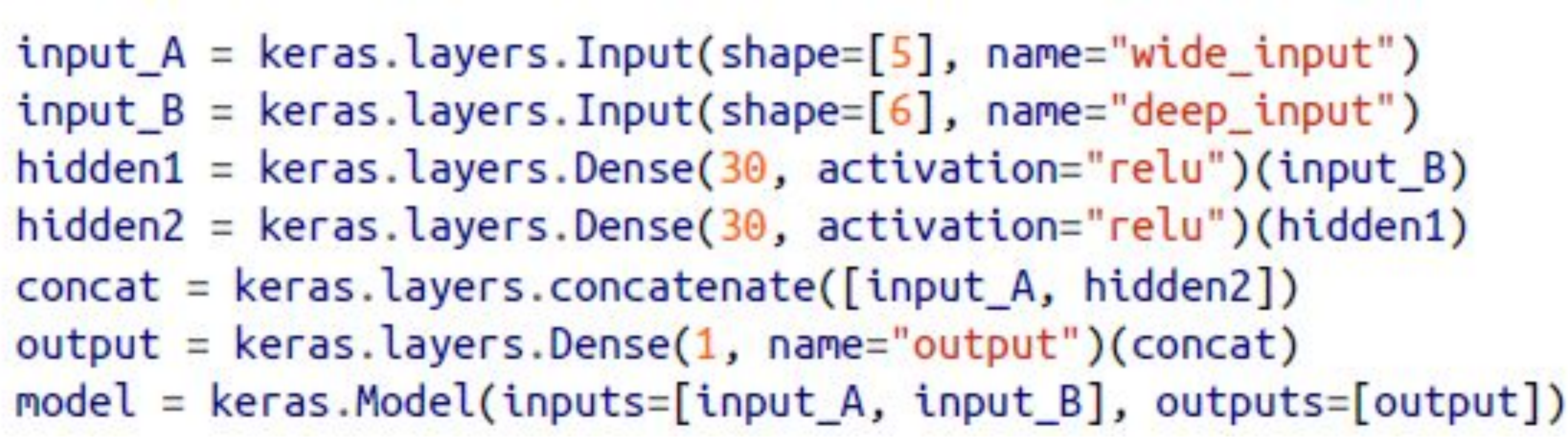


```
input_ = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input_)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.Concatenate()([input_, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.Model(inputs=[input_], outputs=[output])
```

Model using Functional API : Illustration 2

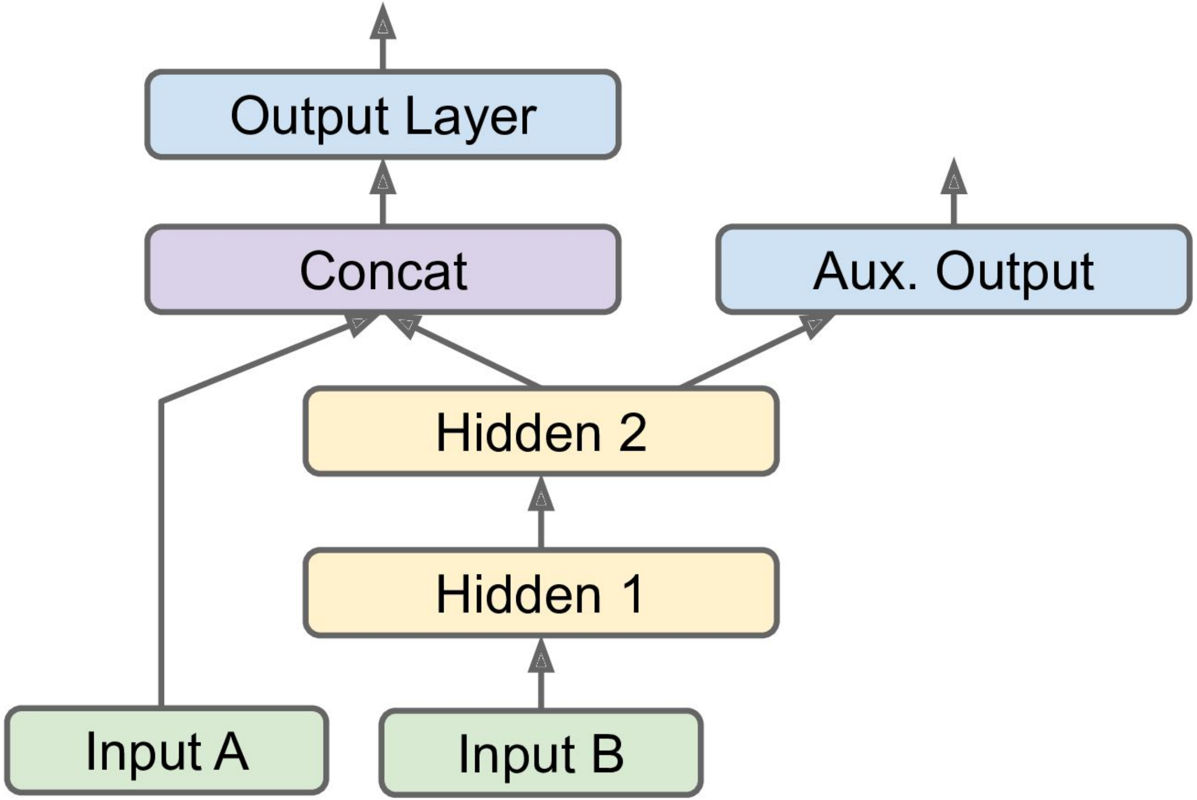


Model using Functional API : Illustration 2

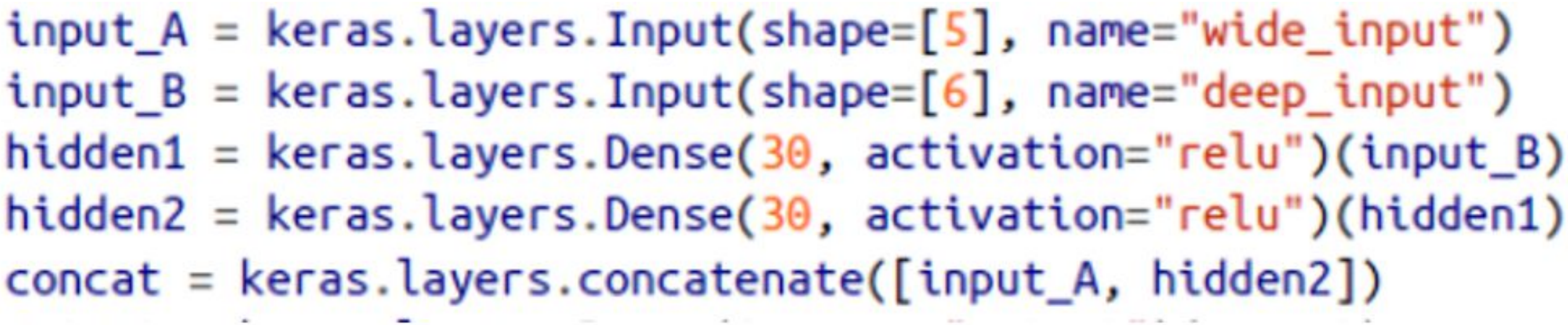


```
input_A = keras.layers.Input(shape=[5], name="wide_input")
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1, name="output")(concat)
model = keras.Model(inputs=[input_A, input_B], outputs=[output])
```

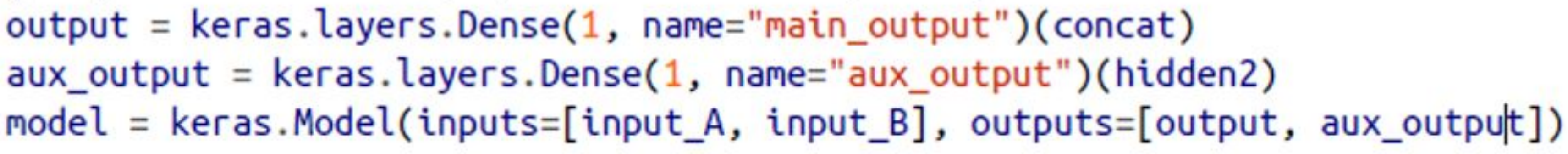
Model using Functional API : Illustration 3 (Auxiliary output)



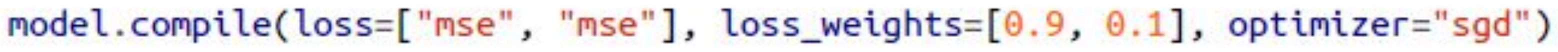
Model using Functional API : Illustration 3 (Auxiliary output)



```
input_A = keras.layers.Input(shape=[5], name="wide_input")
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
```



```
output = keras.layers.Dense(1, name="main_output")(concat)
aux_output = keras.layers.Dense(1, name="aux_output")(hidden2)
model = keras.Model(inputs=[input_A, input_B], outputs=[output, aux_output])
```



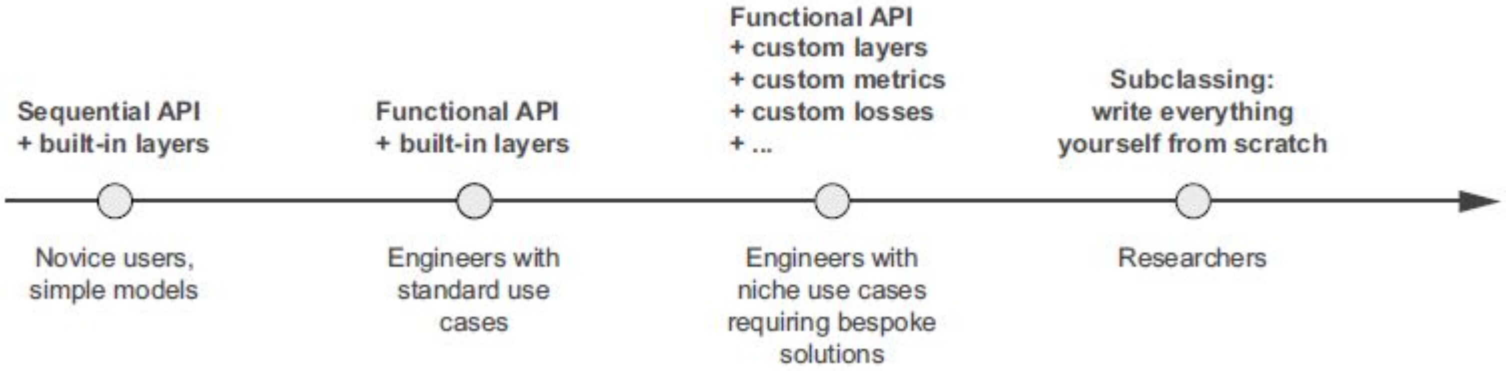
```
model.compile(loss=["mse", "mse"], loss_weights=[0.9, 0.1], optimizer="sgd")
```

Sequential & Functional APIs

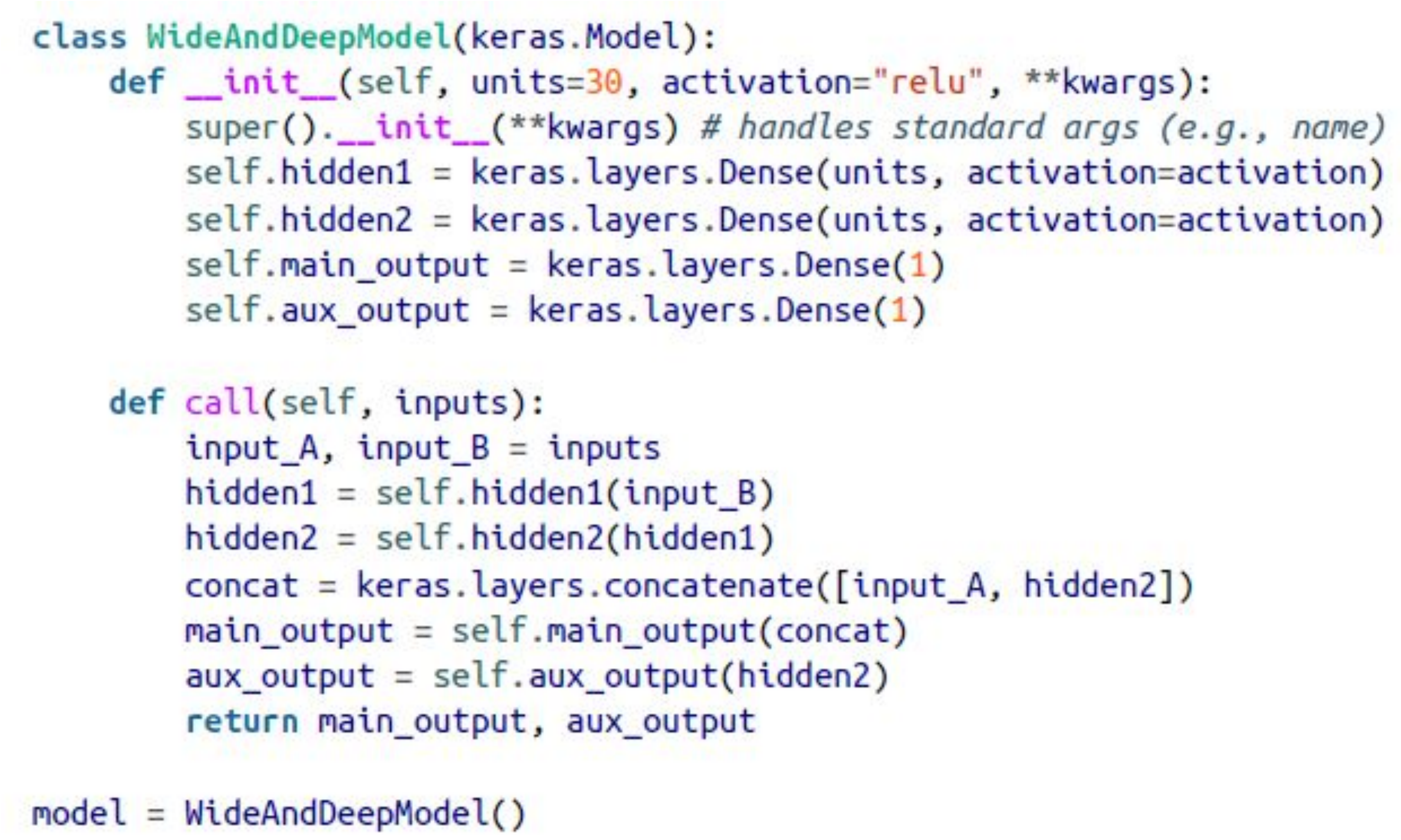
- Both the Sequential API and the Functional API are declarative
- Start modeling ANNs by declaring which layers to use and how they should be connected, and only then model takes some data for training or inference.
- Advantages of declarative approach:
  - the model can easily be saved, cloned, and shared;
  - The model structure can be displayed and analyzed;
  - the framework can infer shapes and check types, so errors can be caught early (i.e., before any data ever goes through the model).
  - Easy to debug, since the whole model is a static graph of layers.
- Disadvantages of declarative approach:
  - It iss static.
  - Some models involve loops, varying shapes, conditional branching, and expect other dynamic behaviors.
- To address such limitations, the Subclassing API is required

Model using Subclassing API

- A low-level option to write and build everything from scratch.
- Ideal when the model developers want full control over every little thing.
- Imposes limitations to access to many built-in Keras features, and increases the risk of making mistakes.



Model using Subclassing API



```python
class WideAndDeepModel(keras.Model):
    def __init__(self, units=30, activation="relu", **kwargs):
        super().__init__(**kwargs) # handles standard args (e.g., name)
        self.hidden1 = keras.layers.Dense(units, activation=activation)
        self.hidden2 = keras.layers.Dense(units, activation=activation)
        self.main_output = keras.layers.Dense(1)
        self.aux_output = keras.layers.Dense(1)

    def call(self, inputs):
        input_A, input_B = inputs
        hidden1 = self.hidden1(input_B)
        hidden2 = self.hidden2(hidden1)
        concat = keras.layers.concatenate([input_A, hidden2])
        main_output = self.main_output(concat)
        aux_output = self.aux_output(hidden2)
        return main_output, aux_output

model = WideAndDeepModel()
```