# The Sequential model

**Author:** fchollet
**Date created:** 2020/04/12
**Last modified:** 2020/04/12
**Description:** Complete guide to the Sequential model.

View in Colab • GitHub source

## Setup

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

## When to use a Sequential model

A `Sequential` model is appropriate for **a plain stack of layers** where each layer has **exactly one input tensor and one output tensor**.

Schematically, the following `Sequential` model:

```
# Define Sequential model with 3 layers
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
# Call model on a test input
x = tf.ones((3, 3))
y = model(x)
```

is equivalent to this function:

```
# Create 3 layers
layer1 = layers.Dense(2, activation="relu", name="layer1")
layer2 = layers.Dense(3, activation="relu", name="layer2")
layer3 = layers.Dense(4, name="layer3")

# Call layers on a test input
x = tf.ones((3, 3))
y = layer3(layer2(layer1(x)))
```

A Sequential model is **not appropriate** when:

- Your model has multiple inputs or multiple outputs
- Any of your layers has multiple inputs or multiple outputs
- You need to do layer sharing
- You want non-linear topology (e.g. a residual connection, a multi-branch model)

## Creating a Sequential model

You can create a Sequential model by passing a list of layers to the Sequential constructor:

```
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu"),
        layers.Dense(3, activation="relu"),
        layers.Dense(4),
    ]
)
```

Its layers are accessible via the `layers` attribute:

```
model.layers
```

```
[<tensorflow.python.keras.layers.core.Dense at 0x7fbd5f285a00>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fbd5f285c70>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fbd5f285ee0>]
```

You can also create a Sequential model incrementally via the `add()` method:

```
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
```

Note that there's also a corresponding `pop()` method to remove layers: a Sequential model behaves very much like a list of layers.

```
model.pop()
print(len(model.layers))  # 2
```

```
2
```

Also note that the Sequential constructor accepts a `name` argument, just like any layer or model in Keras. This is useful to annotate TensorBoard graphs with semantically meaningful names.

```
model = keras.Sequential(name="my_sequential")
model.add(layers.Dense(2, activation="relu", name="layer1"))
model.add(layers.Dense(3, activation="relu", name="layer2"))
model.add(layers.Dense(4, name="layer3"))
```

## Specifying the input shape in advance

Generally, all layers in Keras need to know the shape of their inputs in order to be able to create their weights. So when you create a layer like this, initially, it has no weights:

```
layer = layers.Dense(3)
layer.weights  # Empty
```

```
[]
```

It creates its weights the first time it is called on an input, since the shape of the weights depends on the shape of the inputs:

```
# Call layer on a test input
x = tf.ones((1, 4))
y = layer(x)
layer.weights  # Now it has weights, of shape (4, 3) and (3,)
```

```
[<tf.Variable 'dense_6/kernel:0' shape=(4, 3) dtype=float32, numpy=
 array([[-0.5312456 , -0.02559239, -0.77284306],
        [-0.18156391,  0.7774476 , -0.05044252],
        [-0.3559971 ,  0.43751895,  0.3434813 ],
        [-0.25133908,  0.8889308 , -0.6510118 ]], dtype=float32)>,
 <tf.Variable 'dense_6/bias:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>]
```

Naturally, this also applies to Sequential models. When you instantiate a Sequential model without an input shape, it isn't "built": it has no weights (and calling `model.weights` results in an error stating just this). The weights are created when the model first sees some input data:

```
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu"),
        layers.Dense(3, activation="relu"),
        layers.Dense(4),
    ]
)  # No weights at this stage!

# At this point, you can't do this:
# model.weights

# You also can't do this:
# model.summary()

# Call the model on a test input
x = tf.ones((1, 4))
y = model(x)
print("Number of weights after calling the model:", len(model.weights))  # 6
```

```
Number of weights after calling the model: 6
```

Once a model is "built", you can call its `summary()` method to display its contents:

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (1, 2)                    10
_________________________________________________________________
dense_8 (Dense)              (1, 3)                    9
_________________________________________________________________
dense_9 (Dense)              (1, 4)                    16
=================================================================
Total params: 35
Trainable params: 35
Non-trainable params: 0
_________________________________________________________________
```

However, it can be very useful when building a Sequential model incrementally to be able to display the summary of the model so far, including the current output shape. In this case, you should start your model by passing an `Input` object to your model, so that it knows its input shape from the start:

```
model = keras.Sequential()
model.add(keras.Input(shape=(4,)))
model.add(layers.Dense(2, activation="relu"))

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 2)                 10
=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
_________________________________________________________________
```

Note that the `Input` object is not displayed as part of `model.layers`, since it isn't a layer:

```
model.layers
```

```
[<tensorflow.python.keras.layers.core.Dense at 0x7fbd5f1776d0>]
```

A simple alternative is to just pass an `input_shape` argument to your first layer:

```
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu", input_shape=(4,)))

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 2)                 10
=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
_________________________________________________________________
```

Models built with a predefined input shape like this always have weights (even before seeing any data) and always have a defined output shape.

In general, it's a recommended best practice to always specify the input shape of a Sequential model in advance if you know what it is.

---

## A common debugging workflow: `add()` + `summary()`

When building a new Sequential architecture, it's useful to incrementally stack layers with `add()` and frequently print model summaries. For instance, this enables you to monitor how a stack of `Conv2D` and `MaxPooling2D` layers is downsampling image feature maps:

```
model = keras.Sequential()
model.add(keras.Input(shape=(250, 250, 3)))  # 250x250 RGB images
model.add(layers.Conv2D(32, 5, strides=2, activation="relu"))
model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.MaxPooling2D(3))

# Can you guess what the current output shape is at this point? Probably not.
# Let's just print it:
model.summary()

# The answer was: (40, 40, 32), so we can keep downsampling...

model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.MaxPooling2D(3))
model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.MaxPooling2D(2))

# And now?
model.summary()

# Now that we have 4x4 feature maps, time to apply global max pooling.
model.add(layers.GlobalMaxPooling2D())

# Finally, we add a classification layer.
model.add(layers.Dense(10))
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 123, 123, 32)      2432
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 121, 121, 32)      9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 40, 40, 32)        0
=================================================================
Total params: 11,680
Trainable params: 11,680
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 123, 123, 32)      2432
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 121, 121, 32)      9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 40, 40, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 38, 38, 32)        9248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 36, 36, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 32)        9248
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 32)          9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 32)          0
=================================================================
Total params: 48,672
Trainable params: 48,672
Non-trainable params: 0
_________________________________________________________________
```

Very practical, right?

---

## What to do once you have a model

Once your model architecture is ready, you will want to:

- Train your model, evaluate it, and run inference. See our guide to training & evaluation with the built-in loops
- Save your model to disk and restore it. See our guide to serialization & saving.
- Speed up model training by leveraging multiple GPUs. See our guide to multi-GPU and distributed training.

---

## Feature extraction with a Sequential model

Once a Sequential model has been built, it behaves like a Functional API model. This means that every layer has an `input` and `output` attribute. These attributes can be used to do neat things, like quickly creating a model that extracts the outputs of all intermediate layers in a Sequential model:

```
initial_model = keras.Sequential(
    [
        keras.Input(shape=(250, 250, 3)),
        layers.Conv2D(32, 5, strides=2, activation="relu"),
        layers.Conv2D(32, 3, activation="relu"),
        layers.Conv2D(32, 3, activation="relu"),
    ]
)
feature_extractor = keras.Model(
    inputs=initial_model.inputs,
    outputs=[layer.output for layer in initial_model.layers],
)

# Call feature extractor on test input.
x = tf.ones((1, 250, 250, 3))
features = feature_extractor(x)
```

Here's a similar example that only extract features from one layer:

```
initial_model = keras.Sequential(
    [
        keras.Input(shape=(250, 250, 3)),
        layers.Conv2D(32, 5, strides=2, activation="relu"),
        layers.Conv2D(32, 3, activation="relu", name="my_intermediate_layer"),
        layers.Conv2D(32, 3, activation="relu"),
    ]
)
feature_extractor = keras.Model(
    inputs=initial_model.inputs,
    outputs=initial_model.get_layer(name="my_intermediate_layer").output,
)
# Call feature extractor on test input.
x = tf.ones((1, 250, 250, 3))
features = feature_extractor(x)
```

---

## Transfer learning with a Sequential model

Transfer learning consists of freezing the bottom layers in a model and only training the top layers. If you aren't familiar with it, make sure to read our guide to transfer learning.

Here are two common transfer learning blueprint involving Sequential models.

First, let's say that you have a Sequential model, and you want to freeze all layers except the last one. In this case, you would simply iterate over `model.layers` and set `layer.trainable = False` on each layer, except the last one. Like this:

```
model = keras.Sequential([
    keras.Input(shape=(784)),
    layers.Dense(32, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(10),
])

# Presumably you would want to first load pre-trained weights.
model.load_weights(...)

# Freeze all layers except the last one.
for layer in model.layers[:-1]:
  layer.trainable = False

# Recompile and train (this will only update the weights of the last layer).
model.compile(...)
model.fit(...)
```

Another common blueprint is to use a Sequential model to stack a pre-trained model and some freshly initialized classification layers. Like this:

```
# Load a convolutional base with pre-trained weights
base_model = keras.applications.Xception(
    weights='imagenet',
    include_top=False,
    pooling='avg')

# Freeze the base model
base_model.trainable = False

# Use a Sequential model to add a trainable classifier on top
model = keras.Sequential([
    base_model,
    layers.Dense(1000),
])

# Compile & train
model.compile(...)
model.fit(...)
```

If you do transfer learning, you will probably find yourself frequently using these two patterns.

That's about all you need to know about Sequential models!

To find out more about building models in Keras, see:

- Guide to the Functional API
- Guide to making new Layers & Models via subclassing