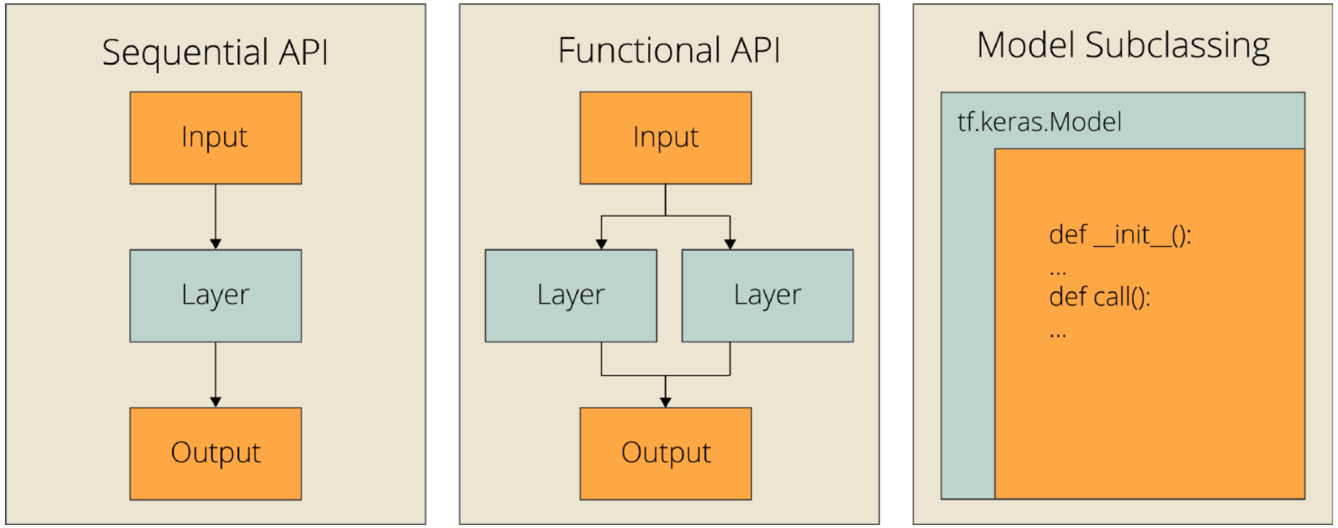
Intro to Neural Nets

Week 5: Functional API

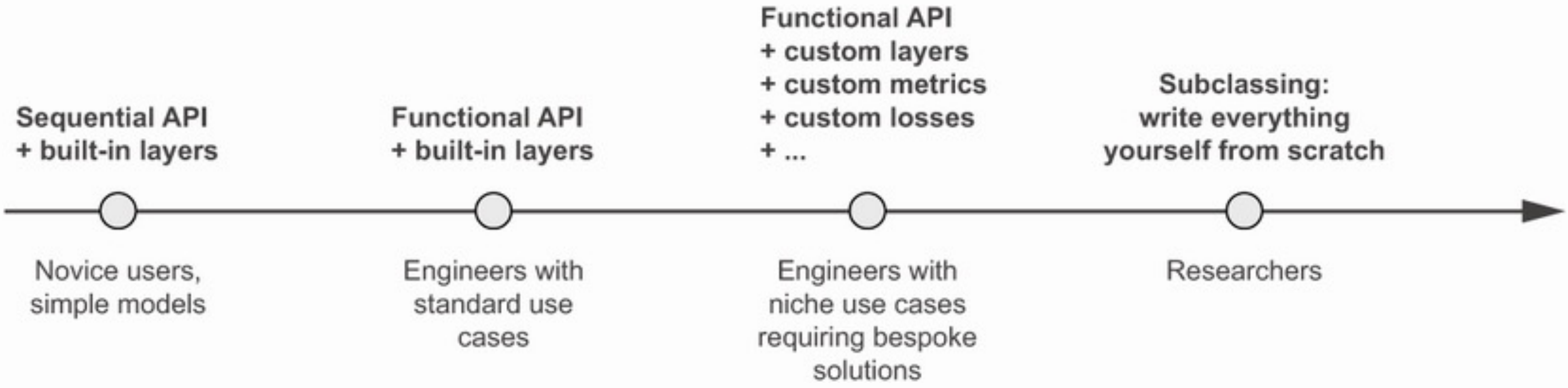
Today's Agenda

Functional vs. Sequential API (vs. Subclassing)

- Creating Keras models with the Sequential class vs. the Functional API
- Custom Loss functions
- Using TensorBoard to monitor training and evaluation metrics
- Writing training and evaluation loops from scratch



Alternative Options /w Keras



Sequential API: Topology

Listing 7.1 The `Sequential` class

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

Same Thing, but Adding Layers Dynamically

```
model = keras.Sequential()
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(10, activation="softmax"))
```

Sequential API: Initialization

Listing 7.3 Models that aren't yet built have no weights

```
>>> model.weights
ValueError: Weights for model sequential_1 have not yet been created.
```

Listing 7.4 Calling a model for the first time to build it

```
>>> model.build(input_shape=(None, 3))
>>> model.weights
[<tf.Variable "dense_2/kernel:0" shape=(3, 64) dtype=float32, ... >,
 <tf.Variable "dense_2/bias:0" shape=(64,) dtype=float32, ... >
 <tf.Variable "dense_3/kernel:0" shape=(64, 10) dtype=float32, ... >,
 <tf.Variable "dense_3/bias:0" shape=(10,) dtype=float32, ... >]
```

Listing 7.7 Specifying the input shape of your model in advance

```
model = keras.Sequential()
model.add(keras.Input(shape=(3,)))
model.add(layers.Dense(64, activation="relu"))
```

Sequential API: Summary

Listing 7.5 The `summary()` method

```
>>> model.summary()
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 64)                256
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 906
Trainable params: 906
Non-trainable params: 0
_________________________________________________________________
```

Functional API: Topology

Define Each Layer and its Linkages (If Any)

- When you specify a given layer, you state what input it takes (if any).
- This lets you build more complicated topologies (e.g., branches, multi-outputs, letting earlier layers feed into later layers in the network, and so on.

Listing 7.8 A simple Functional model with two `Dense` layers

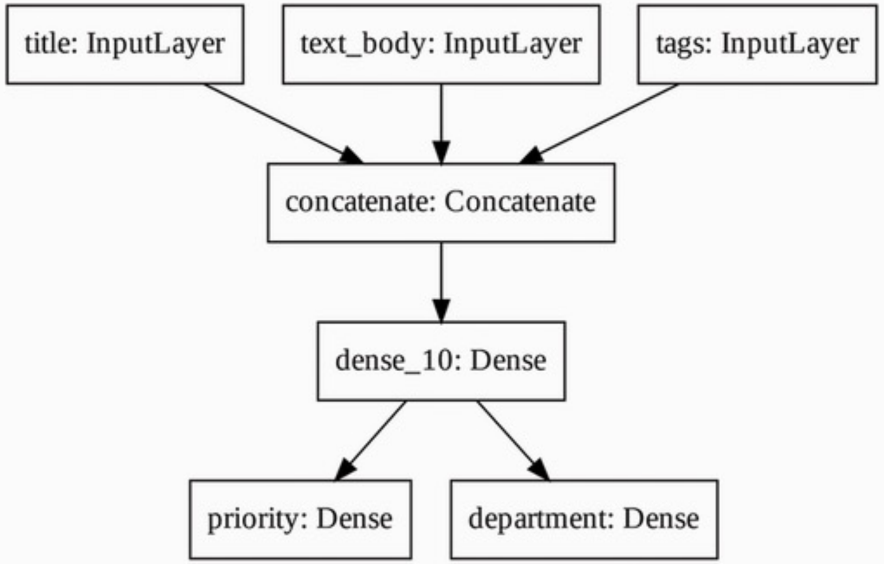
```
inputs = keras.Input(shape=(3,), name="my_input")
features = layers.Dense(64, activation="relu")(inputs)
outputs = layers.Dense(10, activation="softmax")(features)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Complex Topologies

We Can Define a Multi-Input (Branches) to Multi-Output (Labels) Topology

- This because useful for multi-modal data, or for introducing feedback loops, etc. into the network.

```python
keras.utils.plot_model(model, "ticket_classifier.png")
```



Complex Topologies

We Can Define a Multi-Input (Branches) to Multi-Output (Labels) Topology

- This is useful for multi-modal data, residual links, or feedback loops, etc.

Listing 7.9 A multi-input, multi-output Functional model

```
vocabulary_size = 10000
num_tags = 100
num_departments = 4

title = keras.Input(shape=(vocabulary_size,), name="title")
text_body = keras.Input(shape=(vocabulary_size,), name="text_body")
tags = keras.Input(shape=(num_tags,), name="tags")

features = layers.Concatenate()([title, text_body, tags])
features = layers.Dense(64, activation="relu")(features)

priority = layers.Dense(1, activation="sigmoid", name="priority")(features)
department = layers.Dense(
    num_departments, activation="softmax", name="department")(features)

model = keras.Model(inputs=[title, text_body, tags],
                    outputs=[priority, department])
```

Functional API: Training

Pass Lists of Input Arrays, and Specify Lists of Loss Functions / Metrics

- We need to pass in a list of arrays addressing all inputs, as well as a list of labels for all outputs.
- We can also provide a list of loss functions and metrics, one of each for each output label.

Listing 7.10 Training a model by providing lists of input and target arrays

```
import numpy as np

num_samples = 1280

title_data = np.random.randint(0, 2, size=(num_samples, vocabulary_size))
text_body_data = np.random.randint(0, 2, size=(num_samples, vocabulary_size))
tags_data = np.random.randint(0, 2, size=(num_samples, num_tags))

priority_data = np.random.random(size=(num_samples, 1))
department_data = np.random.randint(0, 2, size=(num_samples, num_departments))

model.compile(optimizer="rmsprop",
              loss=["mean_squared_error", "categorical_crossentropy"],
              metrics=[["mean_absolute_error"], ["accuracy"]])
model.fit([title_data, text_body_data, tags_data],
          [priority_data, department_data],
          epochs=1)
model.evaluate([title_data, text_body_data, tags_data],
               [priority_data, department_data])
priority_preds, department_preds = model.predict(
    [title_data, text_body_data, tags_data])
```

Functional API: Training

Pass Lists of Input Arrays, and Specify Lists of Loss Functions / Metrics

- Use dictionaries rather than argument positional indexing (you are less likely to make a mistake this way!)

Listing 7.11 Training a model by providing dicts of input and target arrays

```
model.compile(optimizer="rmsprop",
              loss={"priority": "mean_squared_error", "department":
                    "categorical_crossentropy"},
              metrics={"priority": ["mean_absolute_error"], "department":
                       ["accuracy"]})
model.fit({"title": title_data, "text_body": text_body_data,
           "tags": tags_data},
          {"priority": priority_data, "department": department_data},
          epochs=1)
model.evaluate({"title": title_data, "text_body": text_body_data,
                "tags": tags_data},
               {"priority": priority_data, "department": department_data})
priority_preds, department_preds = model.predict(
    {"title": title_data, "text_body": text_body_data, "tags": tags_data})
```

Functional API: Pre-Trained Weights

We Can Use Pre-Weights From Another Trained Model

- Query the layer of an existing model, to extract its weights / bias term.
- Then feed them into a new model.

Let’s Say We Want to Extract Weights from First Dense Layer…

Listing 7.12 Retrieving the inputs or outputs of a layer in a Functional model

```
>>> model.layers
[<tensorflow.python.keras.engine.input_layer.InputLayer at 0x7fa963f9d358>,
 <tensorflow.python.keras.engine.input_layer.InputLayer at 0x7fa963f9d2e8>,
 <tensorflow.python.keras.engine.input_layer.InputLayer at 0x7fa963f9d470>,
 <tensorflow.python.keras.layers.merge.Concatenate at 0x7fa963f9d860>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa964074390>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa963f9d898>,
 <tensorflow.python.keras.layers.core.Dense at 0x7fa963f95470>]
>>> model.layers[3].input
[<tf.Tensor "title:0" shape=(None, 10000) dtype=float32>,
 <tf.Tensor "text_body:0" shape=(None, 10000) dtype=float32>,
 <tf.Tensor "tags:0" shape=(None, 100) dtype=float32>]
>>> model.layers[3].output
<tf.Tensor "concatenate/concat:0" shape=(None, 20100) dtype=float32>
```

Custom Loss Functions

We Can Write a Custom Loss Function

- Many application-specific situations where we might want a custom metric (or loss function).
- One example I will show you is image-sharpening – MSE or MAE are okay, but there are better metrics for image-wide alignment of pixel values.

Function That Accepts Predicted and Actual as Input

```
def psnrLoss(y_true, y_pred, max_pixel=255.0):
    psnr = tf.image.psnr(y_true,y_pred,max_val=max_pixel)
    return psnr

def ssim_msLoss(y_true, y_pred,max_val=255.0):
    ssim = tf.image.ssim_multiscale(y_true,y_pred,max_val)
    return ssim
```

Custom Loss Functions

We Can Write a Custom Loss Function

- Many application-specific situations where we might want a custom metric (or loss function).
- One example I will show you is image-sharpening – MSE or MAE are okay, but there are better metrics for image-wide alignment of pixel values.

Function That Accepts Predicted and Actual as Input

```
def psnrLoss(y_true, y_pred, max_pixel=255.0):
    psnr = tf.image.psnr(y_true,y_pred,max_val=max_pixel)
    return psnr

def ssim_msLoss(y_true, y_pred,max_val=255.0):
    ssim = tf.image.ssim_multiscale(y_true,y_pred,max_val)
    return ssim
```

Callbacks in the Fit Method

We Can Enforce Some Rules During Fitting

- For example, we can tell the fit method to stop the training process early based on < threshold improvements in our objective function.

Listing 7.19 Using the `callbacks` argument in the `fit()` method

```
callbacks_list = [
    keras.callbacks.EarlyStopping(
        monitor="val_accuracy",
        patience=2,
    ),
    keras.callbacks.ModelCheckpoint(
        filepath="checkpoint_path.keras",
        monitor="val_loss",
        save_best_only=True,
    )
]
model = get_mnist_model()
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
model.fit(train_images, train_labels,
          epochs=10,
          callbacks=callbacks_list,
          validation_data=(val_images, val_labels))
```

Save / Load a Trained Model

We Can Save and Re-load Models with Trained Weights for Later Use

- This enables transfer learning, picking up where we left off…

```
model.save('my_model')
model = keras.models.load_model("my_model.keras")
```

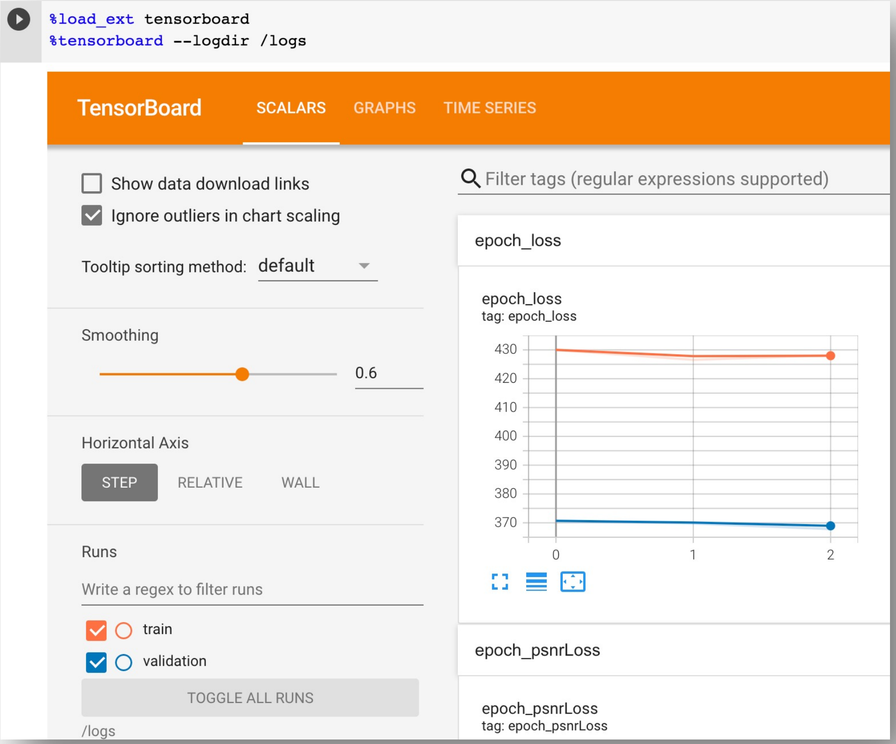
- We might load an old model, extract the first 5 layers, and then stack new layers on the end.
- We might also fix the old model's layers as non-trainable ('calibrate' a new model 'end' on top of the pre-trained model).
- More on this shortly…

Tensorboard

Load in Keras, then add Callback to model.fit()

```
%load_ext tensorboard
%tensorboard --logdir /logs
```

```
,callbacks=keras.callbacks.TensorBoard(log_dir='/logs')
```



Questions?