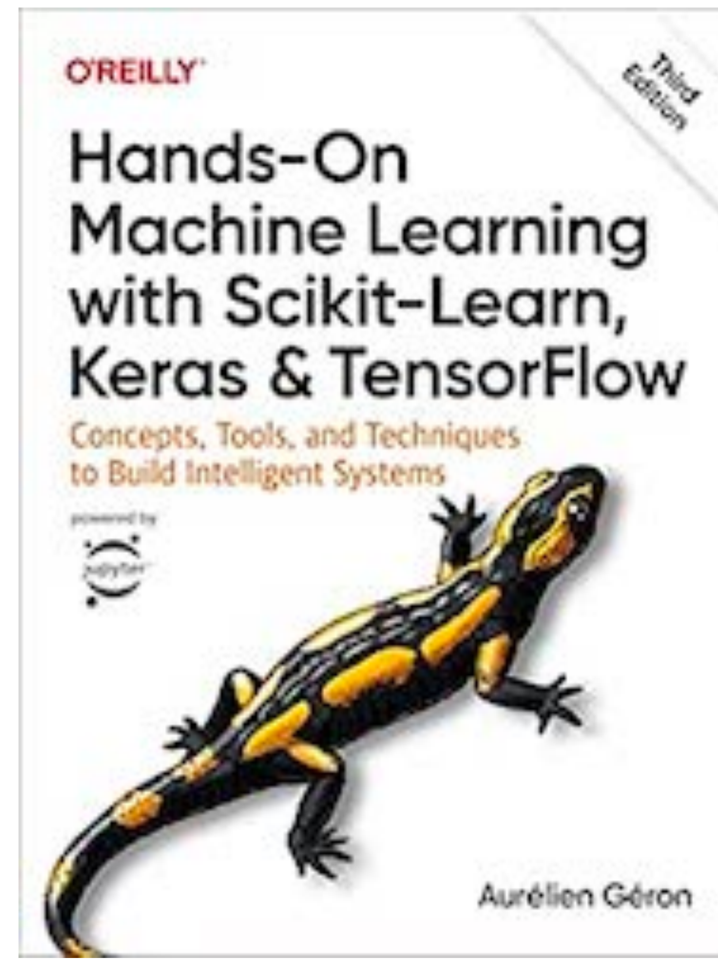
Machine Learning Security

12 Custom Models and Training with Tensorflow



Made Nov 17, 2023

Topics

- A Quick Tour of TensorFlow
- Using TensorFlow like NumPy
- Customizing Models and Training Algorithms
- TensorFlow Functions and Graphs

A Quick Tour of TensorFlow

Keras

- Keras is TensorFlow's high-level API. It can build these neural networks
    - Regression & Classification
    - Wide & Deep nets
    - Nets using batch normalization, dropout, and learning rate schedules
    - These suffice for 95% of use cases

TensorFlow

- TensorFlow's low-level API allows customizing:
  - Loss functions or metrics
  - Layers, models, initializers
  - Regularizers, weight constraints
  - And more

TensorFlow

- A powerful library for numerical computation, including machine learning
- Developed by the Google Brain team
- Powers Google Cloud Speech, Google Photos, and Google Search
- Open-sourced in 2015
- The most widely-used deep-learning library in the industry

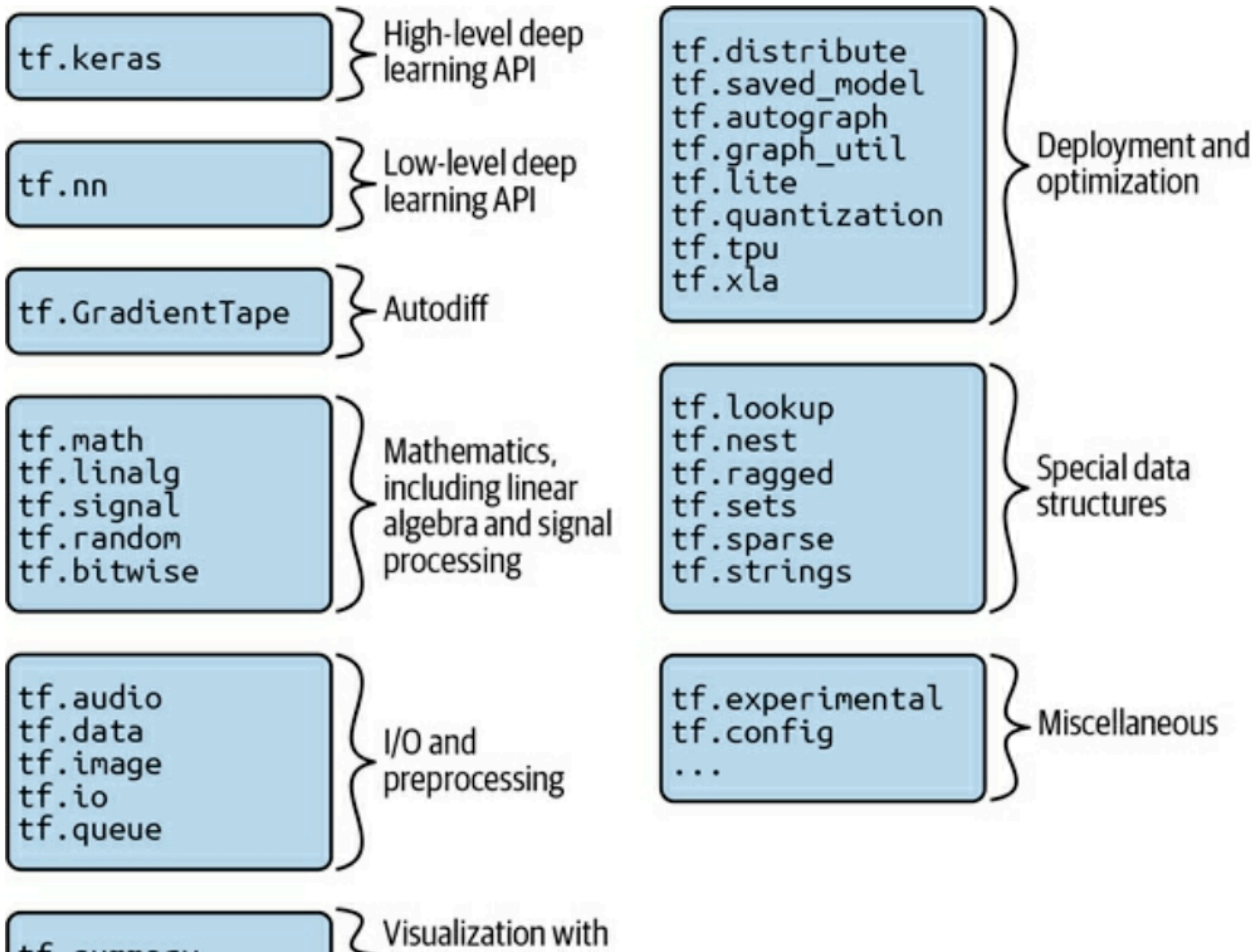
Summary of TensorFlow

- Similar to NumPy, but with GPU support
- Supports distributed computing
- Includes a just-in-time (JIT) compiler
    - Optimizes computations for speed and memory usage
    - Extracts the ***computation graph*** from a Python function
    - Optimizes it and runs it efficiently
- Computation graphs can be exported to other environments
    - Learn in Python on Linux
    - Run in Java on Android

Summary of TensorFlow (continued)

- Implements reverse-mode autodiff
- Provides optimizers like RMSProp and Nadam

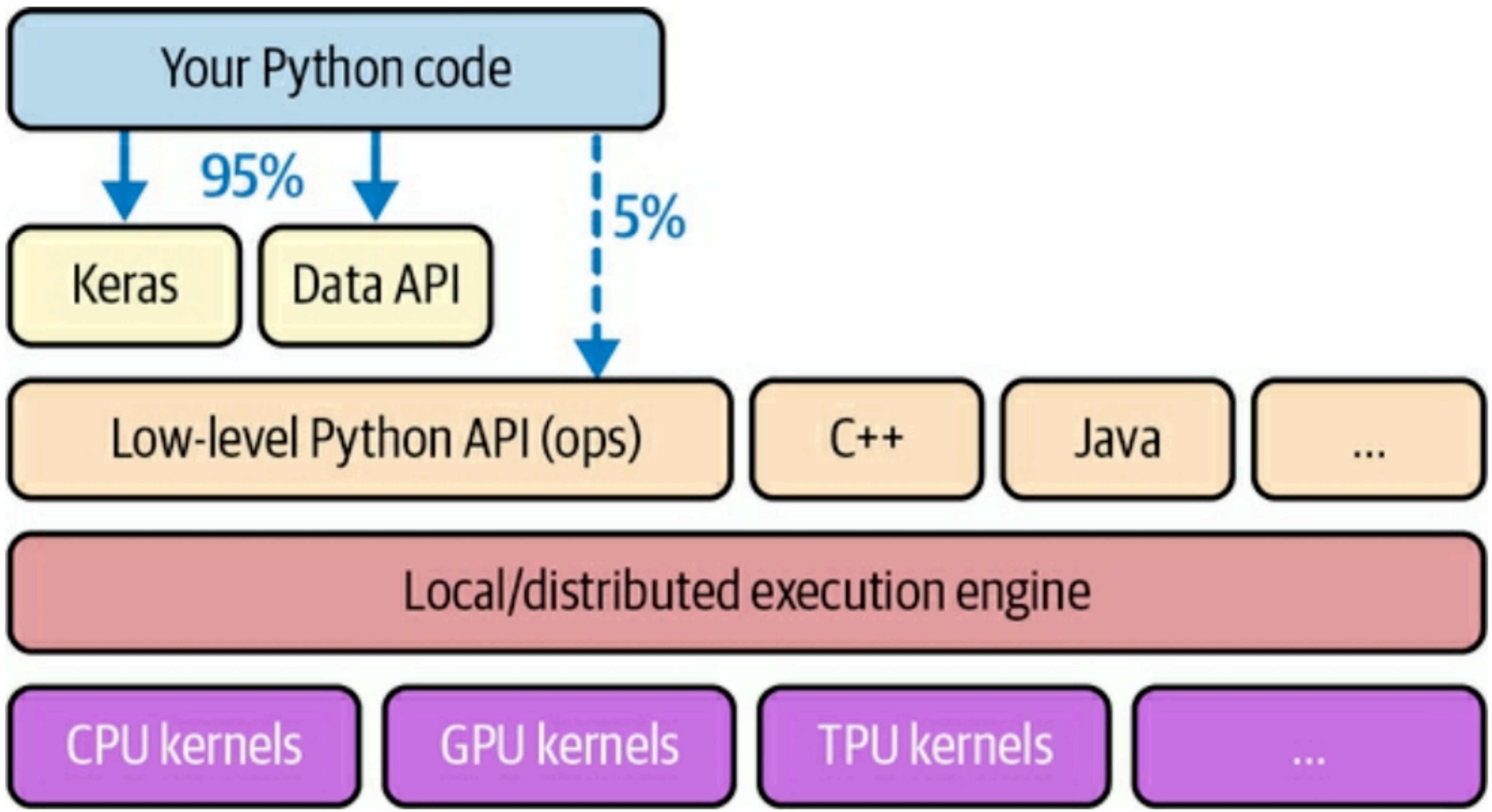
TensorFlow's Python API



Kernels

- Implementations of operations
- Has kernels for CPUs, GPUs, or TPUs (Tensor Processing Units)
  - GPUs can run many threads in parallel
  - TPUs are even faster
    - They are custom ASIC chips for deep learning

Architecture



Other Languages

- Python
- C++
- Java
- Swift
- JavaScript (TensorFlow.js)
  - Can run models directly in a browser

Related Libraries

- TensorBoard for visualization
- TensorFlow Extended (TXD)
    - To productionize TensorFlow projects
    - Data validation, preprocessing, model analysis, and serving
- TensorFlow Hub
    - Easily download and reuse pretrained neural networks

Using TensorFlow like NumPy

Tensors and Operations

- A Tensor is similar to an array

```
import tensorflow as tf
t = tf.constant([[1., 2., 3.], [4., 5., 6.]])
print(t)
```

```
import tensorflow as tf
t = tf.constant([[1., 2., 3.], [4., 5., 6.]])
print(t)
```

```
tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
```

Indexing

```
print(t[0, 0])
print(t[1, 1 ])
print(t[:, 1:])
```

```
tf.Tensor(1.0, shape=(), dtype=float32)
tf.Tensor(5.0, shape=(), dtype=float32)
tf.Tensor(
[[2. 3.]
 [5. 6.]], shape=(2, 2), dtype=float32)
```

Operations

- @ indicates matrix multiplication

```
print(t + 10)
print(tf.square(t))
print(t @ tf.transpose(t))
```

```
tf.Tensor(
[[11. 12. 13.]
 [14. 15. 16.]], shape=(2, 3), dtype=float32)
tf.Tensor(
[[ 1.  4.  9.]
 [16. 25. 36.]], shape=(2, 3), dtype=float32)
tf.Tensor(
[[14. 32.]
 [32. 77.]], shape=(2, 2), dtype=float32)
```

Type Conversions

- It does not automatically convert types

```
print( "1 + 2:", tf.constant(1) + tf.constant(2))
print( "1.0 + 2:", tf.constant(1.0) + tf.constant(2))
```

```
1 + 2: tf.Tensor(3, shape=(), dtype=int32)
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
<ipython-input-7-f481566c1fb3> in <cell line: 2>()
      1 print( "1 + 2:", tf.constant(1) + tf.constant(2))
----> 2 print( "1.0 + 2:", tf.constant(1.0) + tf.constant(2))

1 frames

/usr/local/lib/python3.10/dist-packages/tensorflow/python/framework/ops.py in
raise_from_not_ok_status(e, name)
   5886 def raise_from_not_ok_status(e, name) -> NoReturn:
   5887   e.message += (" name: " + str(name if name is not None else ""))
-> 5888   raise core._status_to_exception(e) from None  # pylint: disable=protected-
access
   5889
   5890

InvalidArgumentError: cannot compute AddV2 as input #1(zero-based) was expected to be
a float tensor but is a int32 tensor [Op:AddV2] name:
```

Cast

- Converts one data type to another

```
[12] print( "1.0 + 2.0:", tf.constant(1.0) + tf.cast( tf.constant(2), dtype=tf.float32 ) )

     1.0 + 2.0: tf.Tensor(3.0, shape=(), dtype=float32)
```

Tensors are Immutable

```python
t = tf.constant([[1., 2., 3.], [4., 5., 6.]])
print(t)
t[0, 0].assign(3)
print(t)
```

```
tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-16-2c51df099382> in <cell line: 3>()
      1 t = tf.constant([[1., 2., 3.], [4., 5., 6.]])
      2 print(t)
----> 3 t[0, 0].assign(3)
      4 print(t)

/usr/local/lib/python3.10/dist-packages/tensorflow/python/framework/tensor.py in
__getattr__(self, name)
    259         tf.experimental.numpy.experimental_enable_numpy_behavior()
    260       """)
--> 261     self.__getattribute__(name)
    262 
    263   @property

AttributeError: 'tensorflow.python.framework.ops.EagerTensor' object has no attribute 'assign'
```

Variables

```
t = tf.Variable([[1., 2., 3.], [4., 5., 6.]])
print(t)
t[0, 0].assign(3)
print(t)
```

```
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[1., 2., 3.],
       [4., 5., 6.]], dtype=float32)>
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[3., 2., 3.],
       [4., 5., 6.]], dtype=float32)>
```

Other Data Structures

- **Sparse tensors**
  - Contains mostly zeroes
- **Tensor arrays**
  - A list of tensors
- **Ragged tensors**
  - Lists of tensors with varying sizes along certain dimensions called the *ragged dimensions*
- **String tensors**
  - Contain byte strings (not Unicode)

Other Data Structures (continued)

- **Sets**
  - A Python data type that does not allow duplicates
  - Are represented as regular tensors
- **Queues**
  - Store tensors across multiple sets, including
    - First-in, first out (FIFO)
    - Queues that prioritize some items
    - Shuffle items
    - Batch items of different sizes with padding

Kahoot!

Ch 12a

Customizing Models and Training Algorithms

Custom Loss Functions

- Huber Loss combines mean squared error and absolute error
- It's available in Keras
- This code demonstrates how to build it yourself

```python
def huber_fn(y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) < 1
    squared_loss = tf.square(error) / 2
    linear_loss  = tf.abs(error) - 0.5
    return tf.where(is_small_error, squared_loss, linear_loss)


model.compile(loss=huber_fn, optimizer="nadam")
model.fit(X_train, y_train, [...])
```

Custom Activation Functions, Initializers, Regularizers, and Constraints

```python
def my_softplus(z):
    return tf.math.log(1.0 + tf.exp(z))

def my_glorot_initializer(shape, dtype=tf.float32):
    stddev = tf.sqrt(2. / (shape[0] + shape[1]))
    return tf.random.normal(shape, stddev=stddev, dtype=dtype)

def my_l1_regularizer(weights):
    return tf.reduce_sum(tf.abs(0.01 * weights))

def my_positive_weights(weights):  # return value is just tf.nn.relu(weights)
    return tf.where(weights < 0., tf.zeros_like(weights), weights)
```

```python
layer = tf.keras.layers.Dense(1, activation=my_softplus,
                              kernel_initializer=my_glorot_initializer,
                              kernel_regularizer=my_l1_regularizer,
                              kernel_constraint=my_positive_weights)
```

Custom Metrics

- **Losses** are used by gradient descent to train a model
    - Must have nonzero gradients
    - May not be easy for humans to interpret
- **Metrics** are used to evaluate a model
    - Must be easy for humans to interpret

```
model.compile(loss="mse", optimizer="nadam", metrics=[create_huber(2.0)])
```

Custom Layers

- Layers without weights, such as Flatten or ReLU, use lambda functions
- This example applies the exponential function to its inputs

```
exponential_layer = tf.keras.layers.Lambda(lambda x: tf.exp(x))
```

Custom Stateful Layers

- With weights
- Create a subclass of the **tf.keras.layers.Layer** class
- This example creates a simplified version of the **Dense** layer

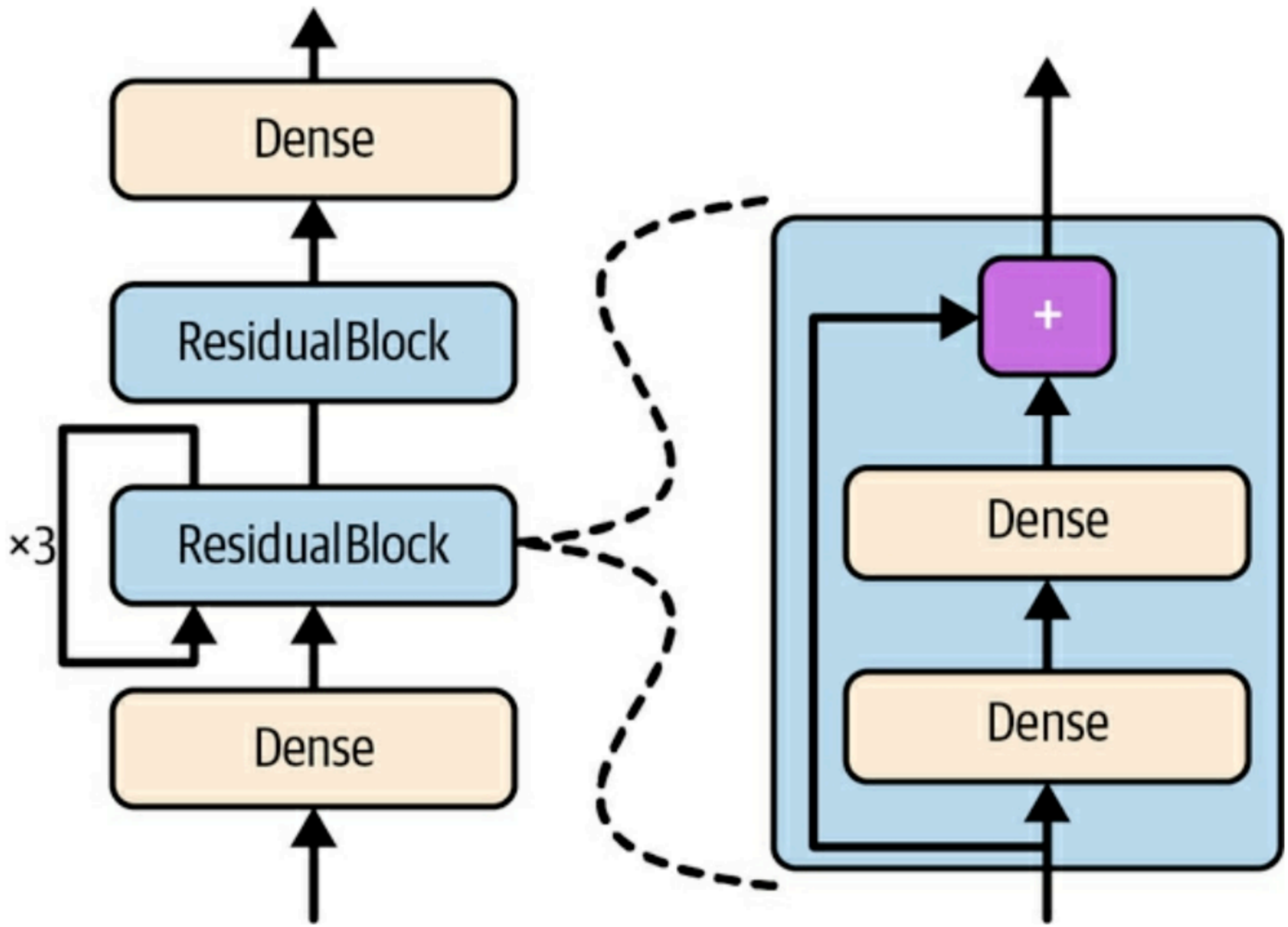
```
class MyDense(tf.keras.layers.Layer):
    def __init__(self, units, activation=None, **kwargs):
        super().__init__(**kwargs)
        self.units = units
        self.activation = tf.keras.activations.get(activation)

    def build(self, batch_input_shape):
        self.kernel = self.add_weight(
            name="kernel", shape=[batch_input_shape[-1], self.units],
            initializer="glorot_normal")
        self.bias = self.add_weight(
            name="bias", shape=[self.units], initializer="zeros")

    def call(self, X):
        return self.activation(X @ self.kernel + self.bias)

    def get_config(self):
        base_config = super().get_config()
        return {**base_config, "units": self.units,
                "activation": tf.keras.activations.serialize(self.activation)}
```

Custom Model Example



ResidualBlock Layer

```
class ResidualBlock(tf.keras.layers.Layer):
    def __init__(self, n_layers, n_neurons, **kwargs):
        super().__init__(**kwargs)
        self.hidden = [tf.keras.layers.Dense(n_neurons, activation="relu",
                                             kernel_initializer="he_normal")
                       for _ in range(n_layers)]

    def call(self, inputs):
        Z = inputs
        for layer in self.hidden:
            Z = layer(Z)
        return inputs + Z
```

ResidualRegressor Model

```
class ResidualRegressor(tf.keras.Model):
    def __init__(self, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.hidden1 = tf.keras.layers.Dense(30, activation="relu",
                                             kernel_initializer="he_normal")
        self.block1 = ResidualBlock(2, 30)
        self.block2 = ResidualBlock(2, 30)
        self.out = tf.keras.layers.Dense(output_dim)

    def call(self, inputs):
        Z = self.hidden1(inputs)
        for _ in range(1 + 3):
            Z = self.block1(Z)
        Z = self.block2(Z)
        return self.out(Z)
```

Losses and Metrics Based on Model Internals

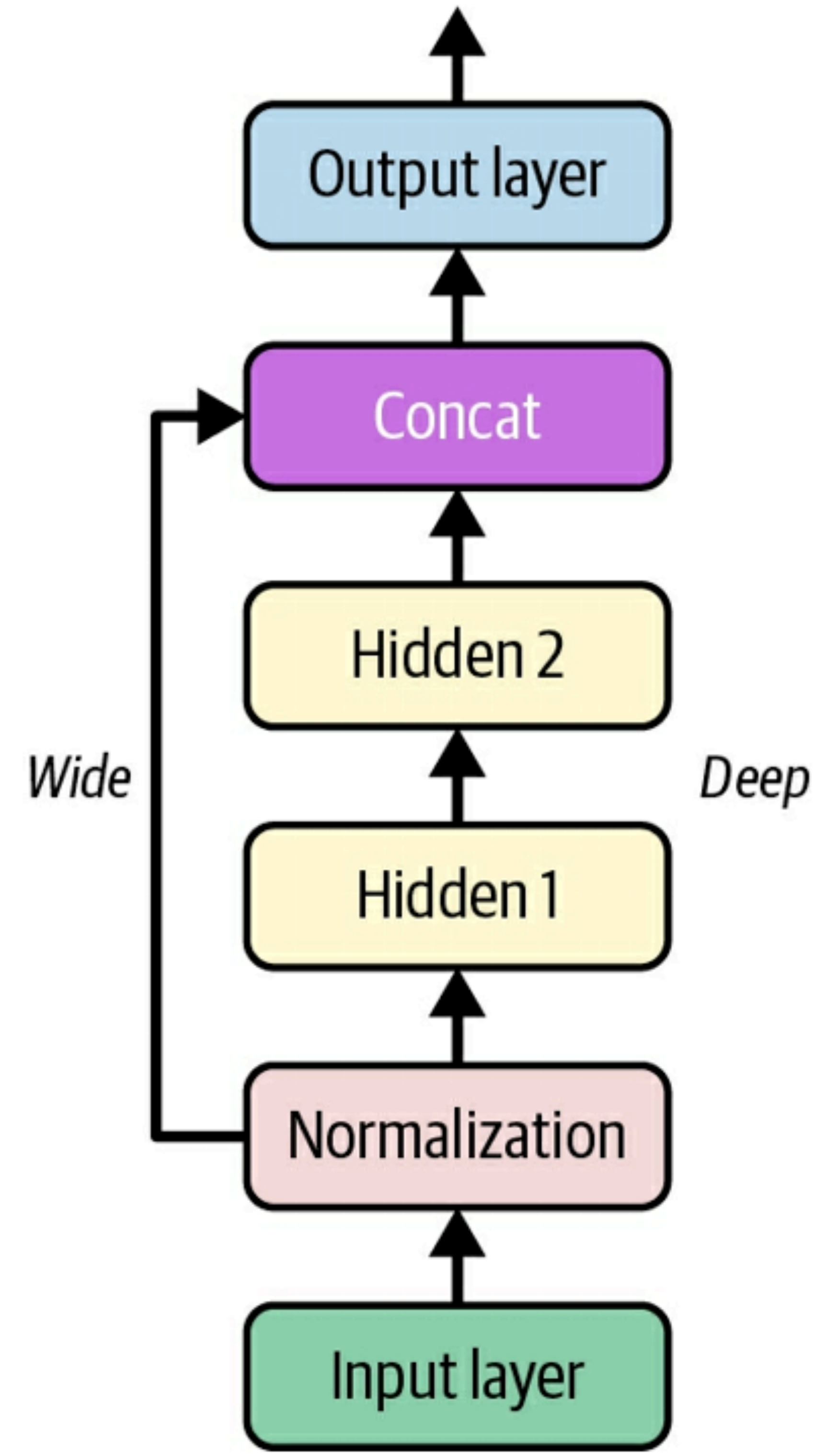
- Losses are normally based on labels and predictions
- You can define losses based on other parts of the model, such as
  - Weights or activations of hidden layers
- This may be useful for regularization or to monitor some internal aspect of your model

Computing Gradients Using Autodiff

- Reverse-mode autodiff computes the gradient using **tf.GradientTape**
    - Tape automatically records every operation involving a variable
    - You can force it to watch other tensors

Custom Training Loops

- Wide & Deep model uses two optimizers
  - One **wide** and one **deep**
- The fit() method uses only one optimizer
- This model requires a custom loop

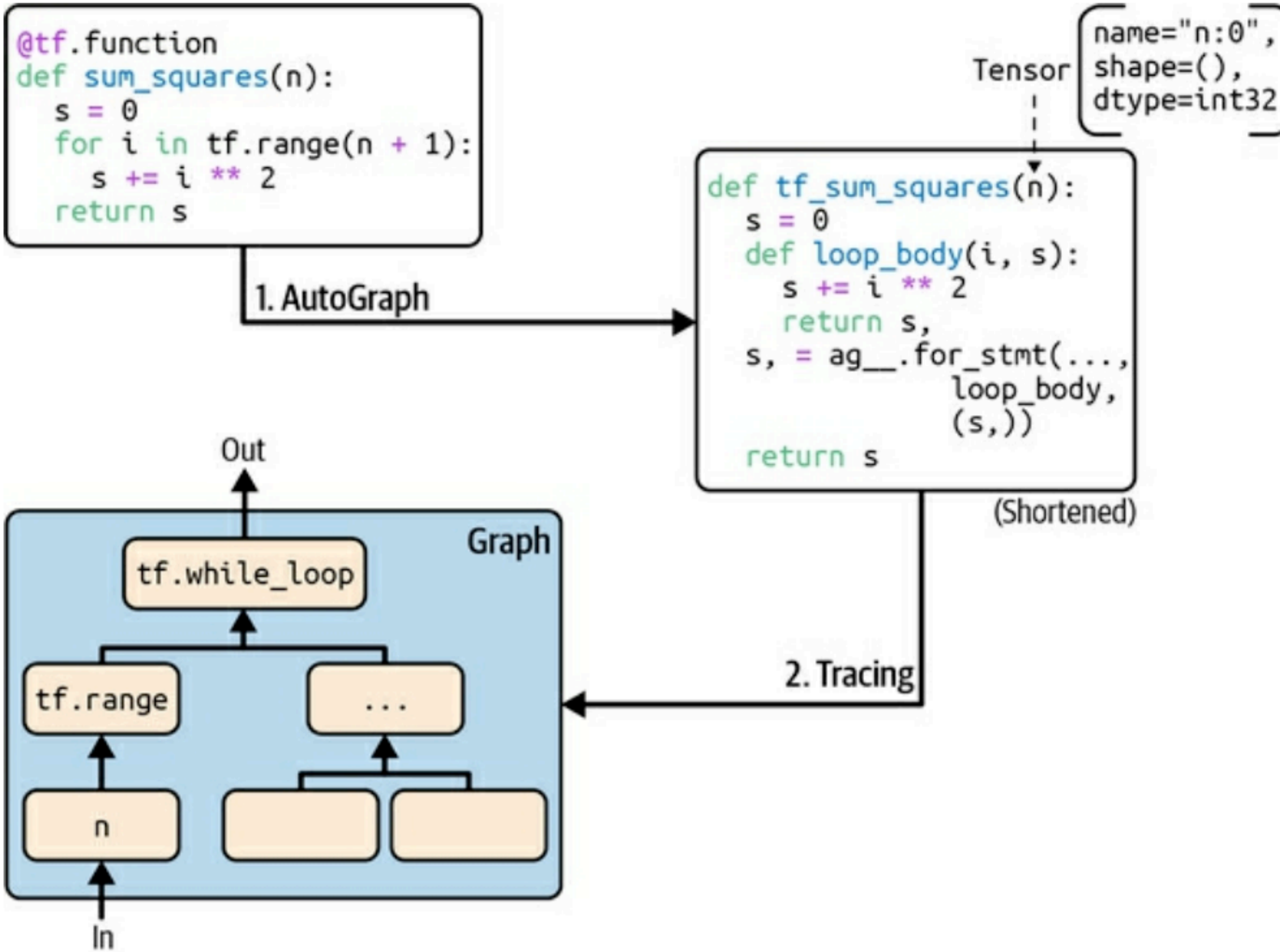


TensorFlow Functions and Graphs

AutoGraph and Tracing

- **AutoGraph**
  - Analyzes Python source code, to
    - capture all the control flow statements, such as
    - **for**, **while**, **if**, **break**, **continue**, **return**
- **Tracing**
  - Runs through the code without performing any calculations
  - To draw the arrows on the graph

Generating Graphs



Kahoot!

Ch 12b