le cnam

Artificial Intelligence and Machine Learning for Connected Industries

Class 12 - TensorFlow

Killian Cressant

Introduction

TensorFlow lower-level API

- 95% of the use cases you will encounter
  - Keras is enough
- It will be useful when we need extra control to write custom
  - Loss functions, metrics, layers, models, initializers, regularizers, weight constraints
  - To fully control the training loop itself
    - Apply special transformations or constraints to the gradients
    - Use multiple optimizers for different parts of the network

TensorFlow

- TensorFlow is a powerful library for numerical computation
  - Particularly well suited and fine-tuned for large-scale Machine Learning
- Developed by the Google Brain team
- It powers many of Google's large-scale services
  - Google Cloud Speech, Google Photos, and Google Search

Functionalities

- Its core is very similar to NumPy but with GPU support
- It supports distributed computing across multiple devices and servers
- It includes a kind of just-in-time (JIT) compiler
  - Allows it to optimize computations for speed and memory usage
- Computation graphs can be exported to a portable format
- It provides some excellent optimizers
  - RMSProp and Nadam
    - Can easily minimize all sorts of loss functions

Functionalities

- Keras
  - tf.keras
- Loading and processing
  - tf.data and tf.io
- Image processing
  - tf.image
- Signal processing
  - tf.signal

TensorFlow’s Python API

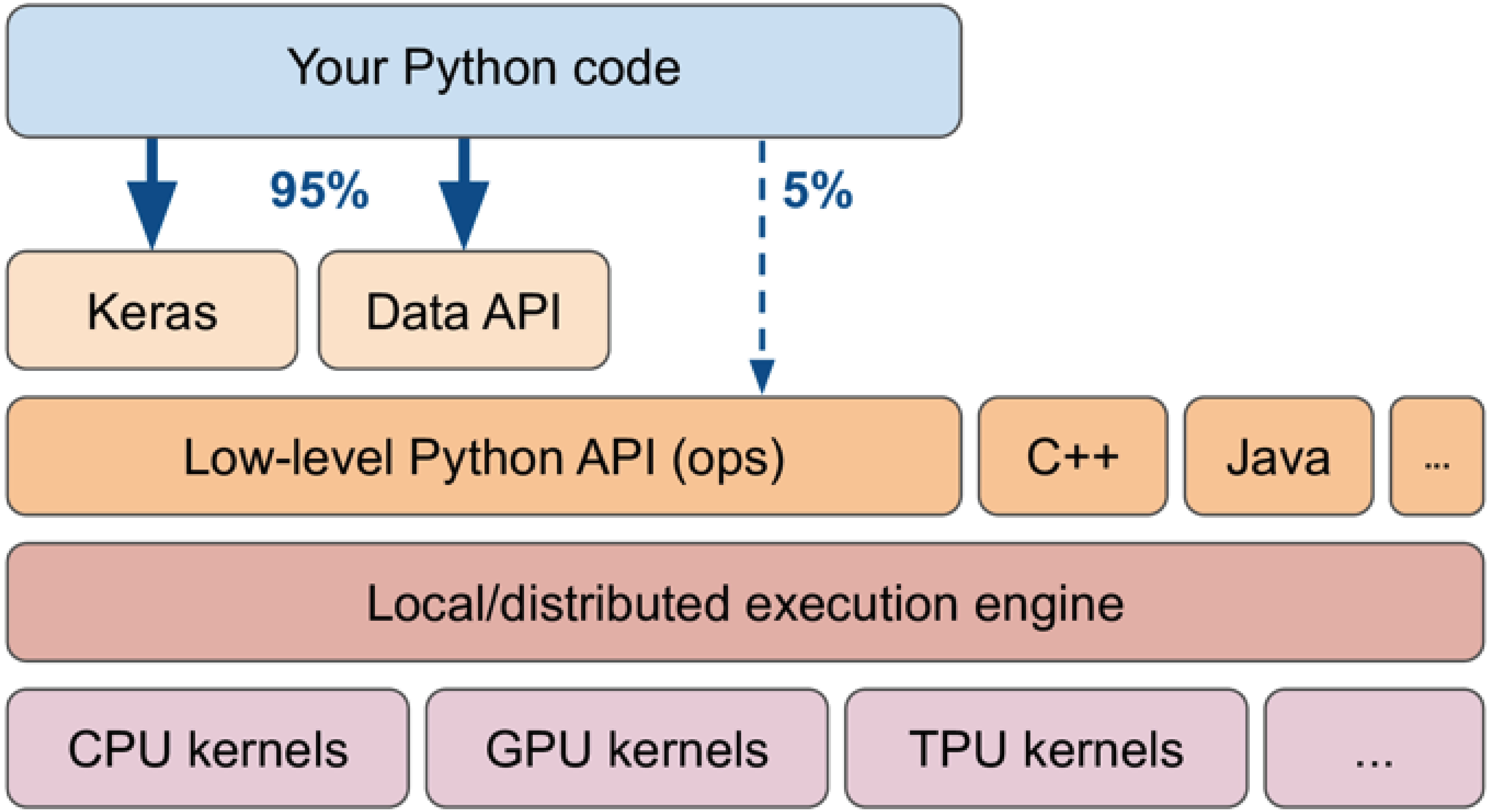
Modules	Description
tf.keras	High-level Deep Learning API
tf.nn	Low-level Deep Learning API
tf.GradientTape	Autodiff
tf.math tf.linalg tf.signal tf.random tf.bitwise	Mathematics, including linear algebra and signal processing
tf.audio tf.data tf.image tf.io tf.queue	I/O and preprocessing
tf.summary	Visualization with TensorBoard
tf.distribute tf.saved_model tf.autograph tf.graph_util tf.lite tf.quantization tf.tpu tf.xla	Deployment and optimization
tf.lookup tf.nest tf.ragged tf.sets tf.sparse tf.strings	Special data structures
tf.experimental tf.config ...	Miscellaneous

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

TensorFlow operations

- At the lowest level
  - Each TensorFlow operation (op for short) is implemented
    - Using highly efficient C++ code
- Many operations have multiple implementations called kernels
  - Each kernel is dedicated to a specific device type
    - CPUs
    - GPUs
    - TPUs —> Tensor Processing Units

TensorFlow architecture



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Multi-platform

- TensorFlow runs on multiple operating systems
  - Windows, Linux, and macOS
  - Mobile devices —> using TensorFlow Lite
    - iOS and Android
- APIs for other languages are also available
  - C++, Java, and Swift APIs
  - JavaScript implementation called TensorFlow.js
    - Makes it possible to run your models directly in your browser

An ecosystem of libraries

- TensorBoard for visualization
- TensorFlow Extended (TFX)
  - A set of libraries built by Google
    - Productionize TensorFlow projects
  - It includes tools for
    - Data validation and preprocessing
    - Model analysis

An ecosystem of libraries

- Google's TensorFlow Hub
  - Easily download and reuse pre-trained neural networks
  - Model garden
    - We can get many neural network architectures
      - Some of them pre-trained

https://paperswithcode.com/

Using TensorFlow like NumPy

TensorFlow like NumPy

- TensorFlow's API revolves around tensors
  - Which flow from operation to operation
  - A tensor is very similar to a NumPy ndarray
    - It is usually a multidimensional array
    - It can also hold a scalar
      - A simple value, such as 27

Tensor example

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

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

```
>>> t.shape
TensorShape([2, 3])
>>> t.dtype
tf.float32
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Indexing

```
>>> t[:, 1:]
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[2., 3.],
       [5., 6.]], dtype=float32)>
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Simple operations

```
>>> t + 10
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[11., 12., 13.],
       [14., 15., 16.]], dtype=float32)>
>>> tf.square(t)
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[ 1.,  4.,  9.],
       [16., 25., 36.]], dtype=float32)>
>>> t @ tf.transpose(t)
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[14., 32.],
       [32., 77.]], dtype=float32)>
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Simple operations

- Most operations are the same in NumPy
- Some have different names
  - tf.reduce_mean() —> np.mean()
  - tf.reduce_sum() —> np.sum()
  - tf.reduce_max() —> np.max()
  - tf.math.log() —> np.log()

Tensors and NumPy

- We can create a tensor from a NumPy array, and vice versa
- We can even apply TensorFlow operations to NumPy arrays
  - And NumPy operations to tensors
- Precision by default
  - NumPy —> uses 64-bit
  - TensorFlow —> 32-bit
    - It is generally more than enough for neural networks
    - It runs faster and uses less RAM

set dtype = tf.float32

Type conversions

- Type conversions can significantly hurt performance
- They can easily go unnoticed when they are done automatically
- TensorFlow does not perform any type conversions automatically
  - It just raises an exception
    - When we try to execute an operation on tensors with incompatible types

Variables

- The tf.Tensor values we've seen so far are immutable
  - This means that we cannot use regular tensors
    - To implement weights in a neural network
      - Since they need to be tweaked by back-propagation
  - Other parameters may also need to change over time

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Variables

```
v.assign(2 * v)            # v now equals [[2., 4., 6.], [8., 10., 12.]]
v[0, 1].assign(42)         # v now equals [[2., 42., 6.], [8., 10., 12.]]
v[:, 2].assign([0., 1.])   # v now equals [[2., 42., 0.], [8., 10., 1.]]
v.scatter_nd_update(       # v now equals [[100., 42., 0.], [8., 10., 200.]]
    indices=[[0, 0], [1, 2]], updates=[100., 200.])
```

- Direct assignment will not work

```
>>> v[1] = [7., 8., 9.]
[...] TypeError: 'ResourceVariable' object does not support item assignment
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Customizing Models and Training Algorithms

Supposition

- Suppose we want to train a regression model
  - Our training set is a bit noisy
- We try to clean up our dataset
  - Eliminating the outliers
    - It is not so efficient
- Which loss function we are going to use?
  - MSE
  - Absolute error

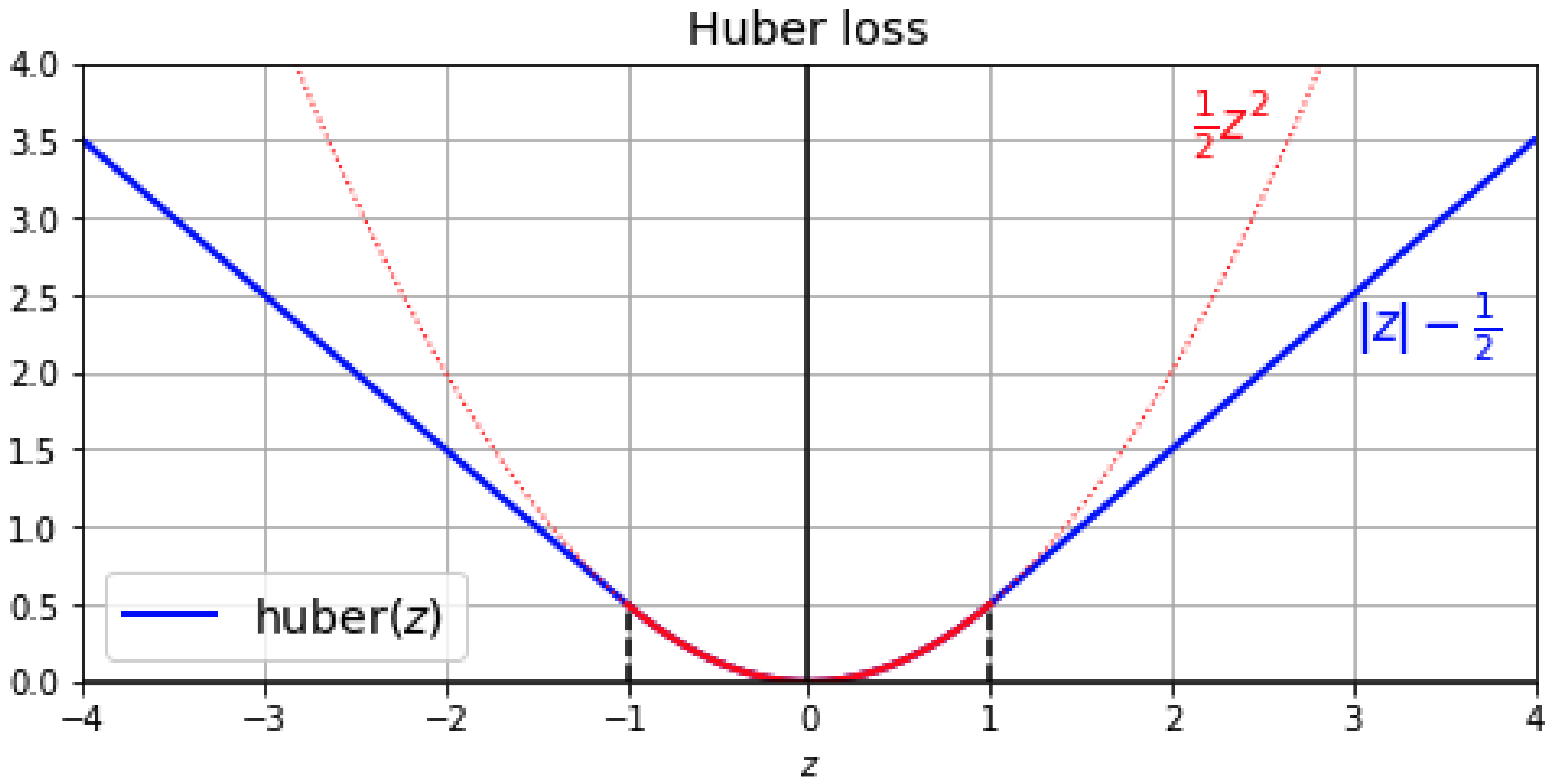
Custom Loss Functions

- Huber loss function
  - Imagine it is not implemented in Keras
    - tf.keras.losses.Huber class

```
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)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Custom Huber loss



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Using our custom loss function

- Now we can use this Huber loss function
  - When we compile the Keras model
    - Train your model as usual

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Saving and loading our model

- When we have a custom function
  - Loading the model
    - We need to provide a dictionary
      - Maps the function name to the actual function

```
model = tf.keras.models.load_model("my_model_with_a_custom_loss",
                                   custom_objects={"huber_fn": huber_fn})
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

- If you decorate the huber_fn() function with
  - @keras.utils.register_keras_serializable()

The threshold as a parameter

```python
def create_huber(threshold=1.0):
    def huber_fn(y_true, y_pred):
        error = y_true - y_pred
        is_small_error = tf.abs(error) < threshold
        squared_loss = tf.square(error) / 2
        linear_loss  = threshold * tf.abs(error) - threshold ** 2 / 2
        return tf.where(is_small_error, squared_loss, linear_loss)
    return huber_fn


model.compile(loss=create_huber(2.0), optimizer="nadam")
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Loading our model

```
model = tf.keras.models.load_model(
    "my_model_with_a_custom_loss_threshold_2",
    custom_objects={"huber_fn": create_huber(2.0)}
)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

An alternative way

```
class HuberLoss(tf.keras.losses.Loss):
    def __init__(self, threshold=1.0, **kwargs):
        self.threshold = threshold
        super().__init__(**kwargs)

    def call(self, y_true, y_pred):
        error = y_true - y_pred
        is_small_error = tf.abs(error) < self.threshold
        squared_loss = tf.square(error) / 2
        linear_loss  = self.threshold * tf.abs(error) - self.threshold**2 /
        return tf.where(is_small_error, squared_loss, linear_loss)

    def get_config(self):
        base_config = super().get_config()
        return {**base_config, "threshold": self.threshold}
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

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))
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Using our custom function

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

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Loading and Preprocessing Data with TensorFlow

Transforming items

- Use the map() method

```
dataset = dataset.map(lambda x: x * 2)  # x is a batch
for item in dataset:
    print(item)
```

```
tf.Tensor([ 0  2  4  6  8 10 12], shape=(7,), dtype=int32)
tf.Tensor([14 16 18  0  2  4  6], shape=(7,), dtype=int32)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

- We can run using multiple threads to speed things up
  - Setting the num_parallel_calls argument to the number of threads to run

Filtering the dataset

- Use the filter() method

```
dataset = dataset.filter(lambda x: tf.reduce_sum(x) > 50)
for item in dataset:
    print(item)
```

```
tf.Tensor([14 16 18  0  2  4  6], shape=(7,), dtype=int32)
tf.Tensor([ 8 10 12 14 16 18  0], shape=(7,), dtype=int32)
tf.Tensor([ 2  4  6  8 10 12 14], shape=(7,), dtype=int32)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Taking a look at some items

- Using the take() method

```
for item in dataset.take(2):
    print(item)
```

```
tf.Tensor([14 16 18  0  2  4  6], shape=(7,), dtype=int32)
tf.Tensor([ 8 10 12 14 16 18  0], shape=(7,), dtype=int32)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Shuffling the dataset

```
dataset = tf.data.Dataset.range(10).repeat(2)
dataset = dataset.shuffle(buffer_size=4, seed=42).batch(7)
for item in dataset:
    print(item)
```

```
tf.Tensor([3 0 1 6 2 5 7], shape=(7,), dtype=int64)
tf.Tensor([8 4 1 9 4 2 3], shape=(7,), dtype=int64)
tf.Tensor([7 5 0 8 9 6], shape=(6,), dtype=int64)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Shuffling the dataset

- For a large dataset that does not fit in memory
  - This simple shuffling-buffer approach may not be sufficient
  - Since the buffer will be small compared to the dataset
- We can divide the dataset into multiple files
  - Multiple files randomly and read them simultaneously
    - Interleaving their records

```
n_readers = 5
dataset = filepath_dataset.interleave(
    lambda filepath: tf.data.TextLineDataset(filepath).skip(1),
    cycle_length=n_readers)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Visualizing some of the dataset

```python
for line in dataset.take(5):
    print(line)
```

```
tf.Tensor(b'4.5909,16.0,[...],33.63,-117.71,2.418', shape=(), dtype=string)
tf.Tensor(b'2.4792,24.0,[...],34.18,-118.38,2.0', shape=(), dtype=string)
tf.Tensor(b'4.2708,45.0,[...],37.48,-122.19,2.67', shape=(), dtype=string)
tf.Tensor(b'2.1856,41.0,[...],32.76,-117.12,1.205', shape=(), dtype=string)
tf.Tensor(b'4.1812,52.0,[...],33.73,-118.31,3.215', shape=(), dtype=string)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Preprocessing the Data

Scaling

```
X_mean, X_std = [...]
n_inputs = 8

def parse_csv_line(line):
    defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
    fields = tf.io.decode_csv(line, record_defaults=defs)
    return tf.stack(fields[:-1]), tf.stack(fields[-1:])

def preprocess(line):
    x, y = parse_csv_line(line)
    return (x - X_mean) / X_std, y
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Scaling

```
X_mean, X_std = [...]
n_inputs = 8

def parse_csv_line(line):
    defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
    fields = tf.io.decode_csv(line, record_defaults=defs)
    return tf.stack(fields[:-1]), tf.stack(fields[-1:])

def preprocess(line):
    x, y = parse_csv_line(line)
    return (x - X_mean) / X_std, y
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Scaling

```
X_mean, X_std = [...]
n_inputs = 8

def parse_csv_line(line):
    defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
    fields = tf.io.decode_csv(line, record_defaults=defs)
    return tf.stack(fields[:-1]), tf.stack(fields[-1:])

def preprocess(line):
    x, y = parse_csv_line(line)
    return (x - X_mean) / X_std, y
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Scaling

```
X_mean, X_std = [...]
n_inputs = 8

def parse_csv_line(line):
    defs = [0.] * n_inputs + [tf.constant([], dtype=tf.float32)]
    fields = tf.io.decode_csv(line, record_defaults=defs)
    return tf.stack(fields[:-1]), tf.stack(fields[-1:])

def preprocess(line):
    x, y = parse_csv_line(line)
    return (x - X_mean) / X_std, y
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

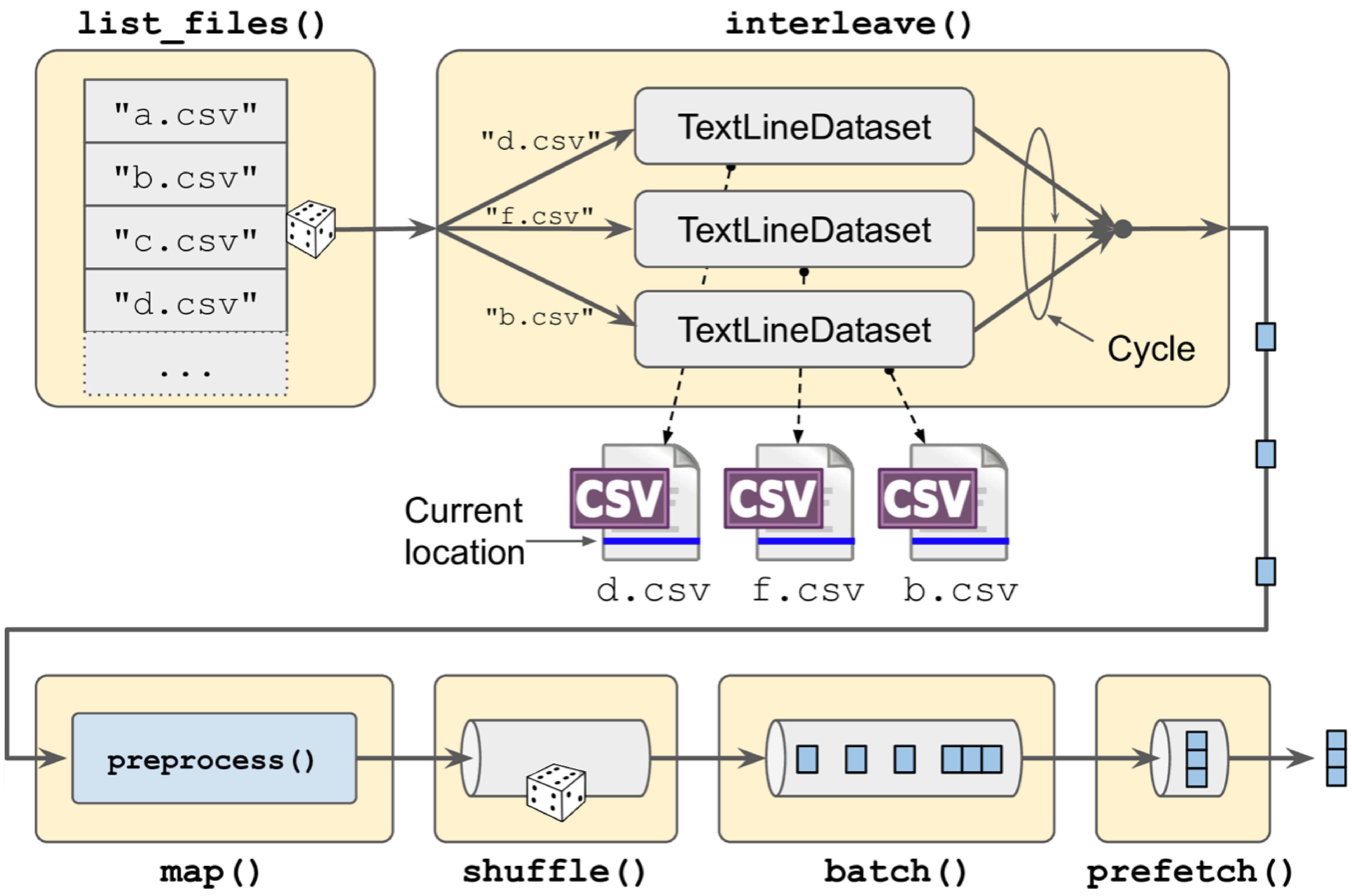
Applying to one line

```
preprocess(b'4.2083,44.0,5.3232,0.9171,846.0,2.3370,37.47,-122.2,2.782')
```

```
(<tf.Tensor: shape=(8,), dtype=float32, numpy=
 array([ 0.16579159,  1.216324  , -0.05204564, -0.39215982, -0.5277444 ,
        -0.2633488 ,  0.8543046 , -1.3072058 ], dtype=float32)>,
 <tf.Tensor: shape=(1,), dtype=float32, numpy=array([2.782], dtype=float32)>
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Loading and preprocessing data from multiple CVS files



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Using with Keras

```python
train_set = csv_reader_dataset(train_filepaths)
valid_set = csv_reader_dataset(valid_filepaths)
test_set = csv_reader_dataset(test_filepaths)
```

```python
model = tf.keras.Sequential([...])
model.compile(loss="mse", optimizer="sgd")
model.fit(train_set, validation_data=valid_set, epochs=5)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Processing layers with Keras

Keras Preprocessing Layers

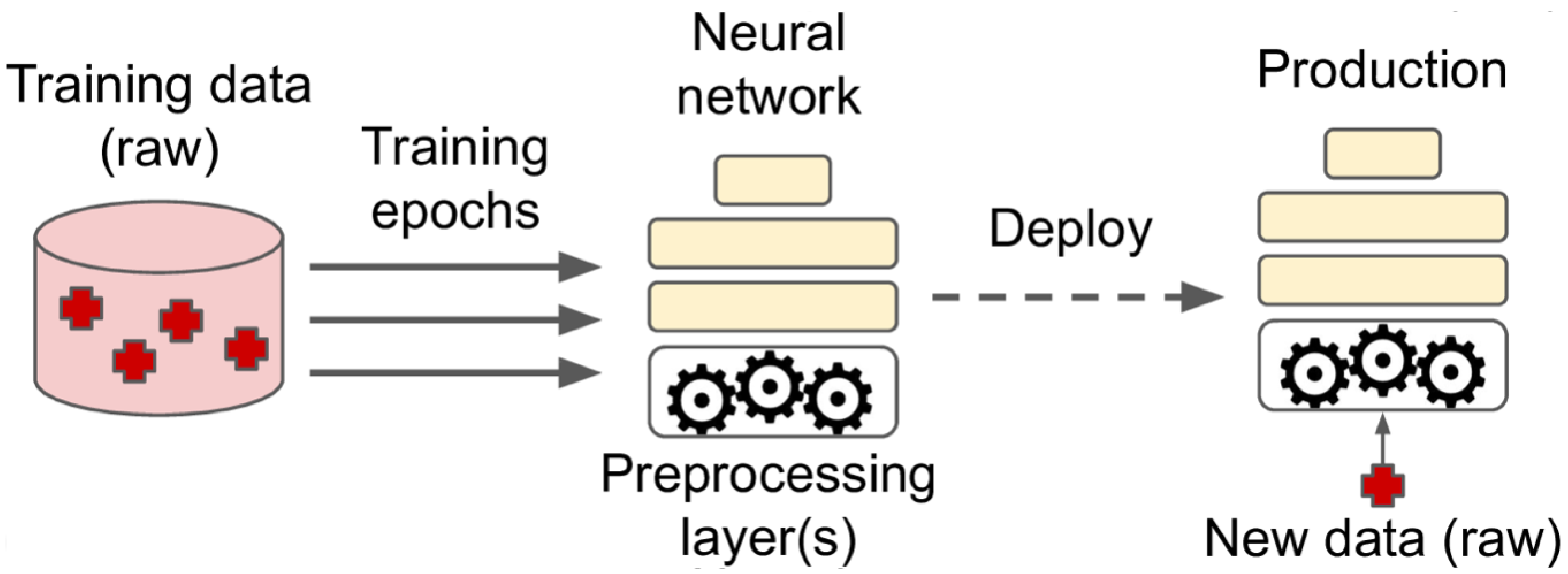
- We can include preprocessing layers directly inside our model
  - It can preprocess all the input data on the fly during training
  - Use the same preprocessing layers in production
- Keras offers many preprocessing layers
  - Can be applied to
    - Numerical features
    - Categorical features
    - Images
    - Text

The Normalization Layer

```
norm_layer = tf.keras.layers.Normalization()
model = tf.keras.models.Sequential([
    norm_layer,
    tf.keras.layers.Dense(1)
])
model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=2e
norm_layer.adapt(X_train)  # computes the mean and variance of every feature
model.fit(X_train, y_train, validation_data=(X_valid, y_valid), epochs=5)
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Including preprocessing layers inside a model



Scaling before training

```
norm_layer = tf.keras.layers.Normalization()
norm_layer.adapt(X_train)
X_train_scaled = norm_layer(X_train)
X_valid_scaled = norm_layer(X_valid)
```

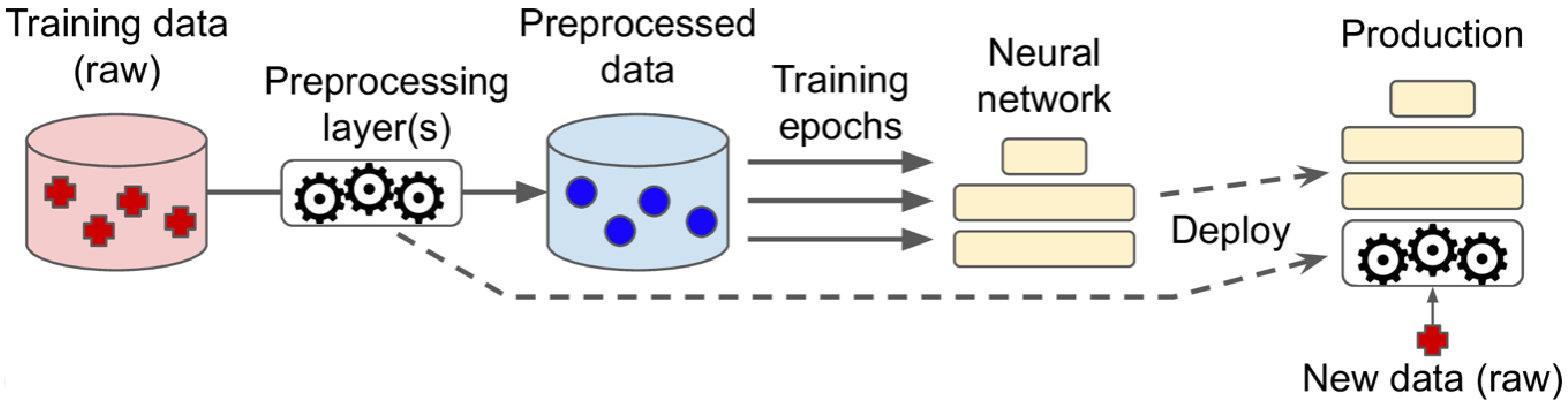
```
model = tf.keras.models.Sequential([tf.keras.layers.Dense(1)])
model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=2e
model.fit(X_train_scaled, y_train, epochs=5,
          validation_data=(X_valid_scaled, y_valid))
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Different final model

```
final_model = tf.keras.Sequential([norm_layer, model])
X_new = X_test[:3]  # pretend we have a few new instances (unscaled)
y_pred = final_model(X_new)  # preprocesses the data and makes predictions
```

Different final model



source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

The Discretization Layer

- Transforms a numerical feature into a categorical feature
  - Mapping value ranges (bins) to categories
- This is sometimes useful for features with
  - Multimodal distributions
  - Have a highly non-linear relationship with the target

Example

```
age = tf.constant([[10.], [93.], [57.], [18.], [37.], [5.]])
discretize_layer = tf.keras.layers.Discretization(bin_boundaries=[18., 5
age_categories = discretize_layer(age)
age_categories
```

```
<tf.Tensor: shape=(6, 1), dtype=int64, numpy=array([[0],[2],[2],[1],[1],[0]]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Another example

- We can provide the number of bins
  - Call the layer's adapt() method
    - Let it find the appropriate bin boundaries
      - Based on the value percentiles

```
discretize_layer = tf.keras.layers.Discretization(num_bins=3)
discretize_layer.adapt(age)
age_categories = discretize_layer(age)
age_categories
```

```
<tf.Tensor: shape=(6, 1), dtype=int64, numpy=array([[1],[2],[2],[1],[2],[0]]
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

The CategoryEncoding Layer

- When there are only a few categories
  - Less than a dozen or two
  - The one-hot encoding is often a good option
- Keras provides the CategoryEncoding layer

One-hot-encoding example

```python
onehot_layer = tf.keras.layers.CategoryEncoding(num_tokens=3)
onehot_layer(age_categories)
```

```
<tf.Tensor: shape=(6, 3), dtype=float32, numpy=
array([[0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.]], dtype=float32)>
```

source: Géron, A. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"

Other layers

- The StringLookup Layer
- The Hashing Layer
- The Embedding Layer
- TextVectorization
- Among others

Artificial Intelligence and Machine Learning for Connected Industries

Class 12 - TensorFlow

Killian Cressant