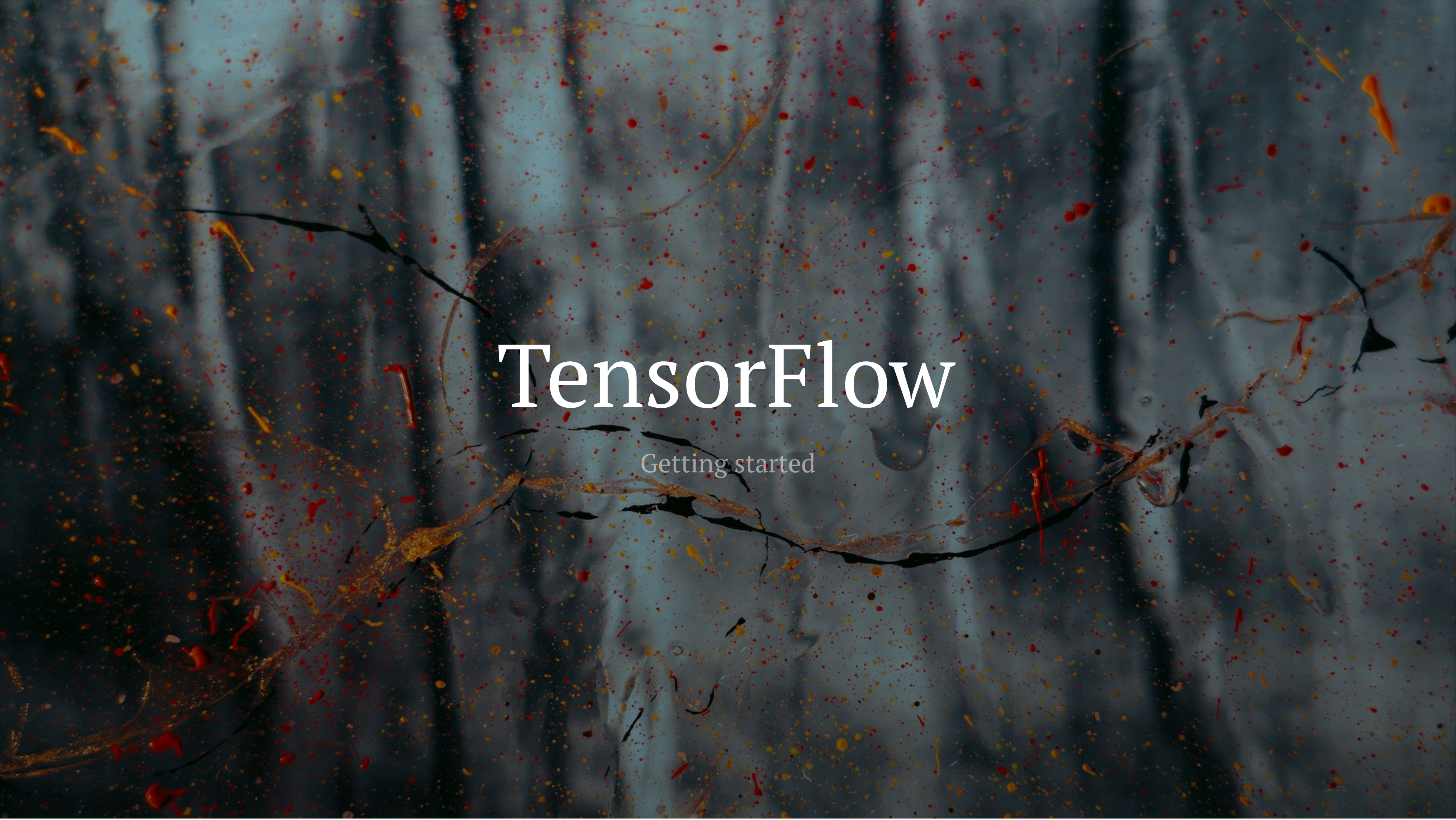


TensorFlow

Getting started

What is TensorFlow?

TensorFlow is a free and open-source software library for machine learning and artificial intelligence

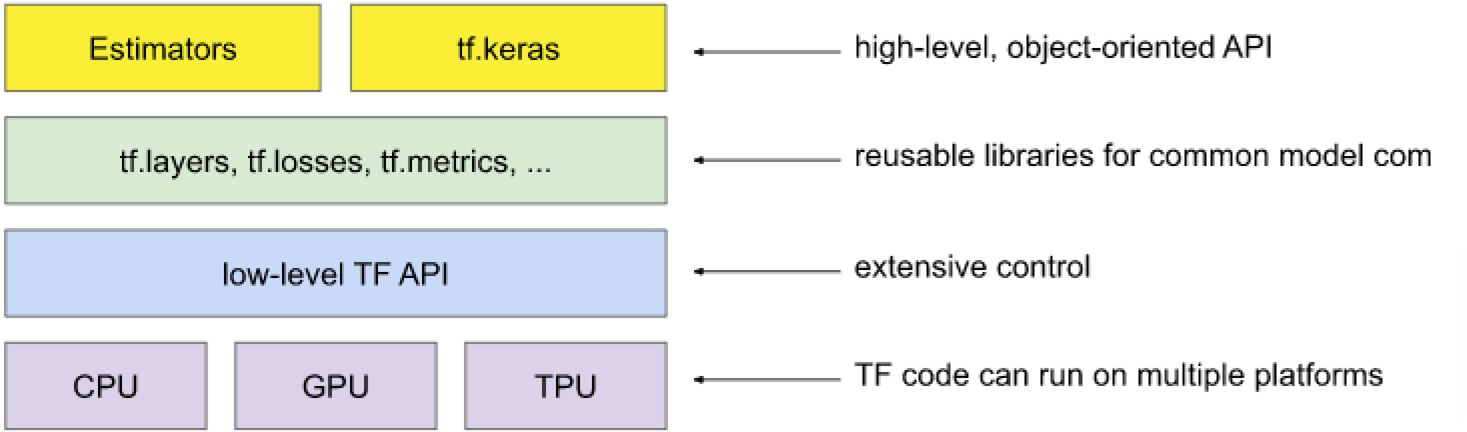
- 🧠 **Developed by Google Brain** - Build for internal Google use in research and production, open-sourced in 2019
- 🐍 **Support most popular languages** - TensorFlow provides a stable Python and C++ API, with non-guaranted backward compatibile API for many others languages
- 👨‍💻 **Developer friendly** - straightforward API, which enables creating even complicated ML models with ease
- 👨‍🏭 **Production ready** - with all it’s variants (TFX, TF Lite, TF.js) allows robust ML production no matter what platform you use
- 🌍 **No setup required** - Google Colab allows creating simple TF models in free, cloud environment
- 📖 **Free courses and resources** - Google and communicty created many great courses to help newcomers start with TF

TensorFlow versions

- TensorFlow - core platform and library for machine learning
- TensorFlow.js - web implementation, allow models to predict on client’s site (in the browser)
- TensorFlow Extended (TFX) - provide componends for building end-to-end production, ex. loading,validating, tuning data.
- TensorFlow Lite (TFLite) - API for mobile and embeddes devices

Getting started with TensorFlow

Library structure



Estimators

tf.keras

high-level, object-oriented API

tf.layers, tf.losses, tf.metrics, ...

reusable libraries for common model com

low-level TF API

extensive control

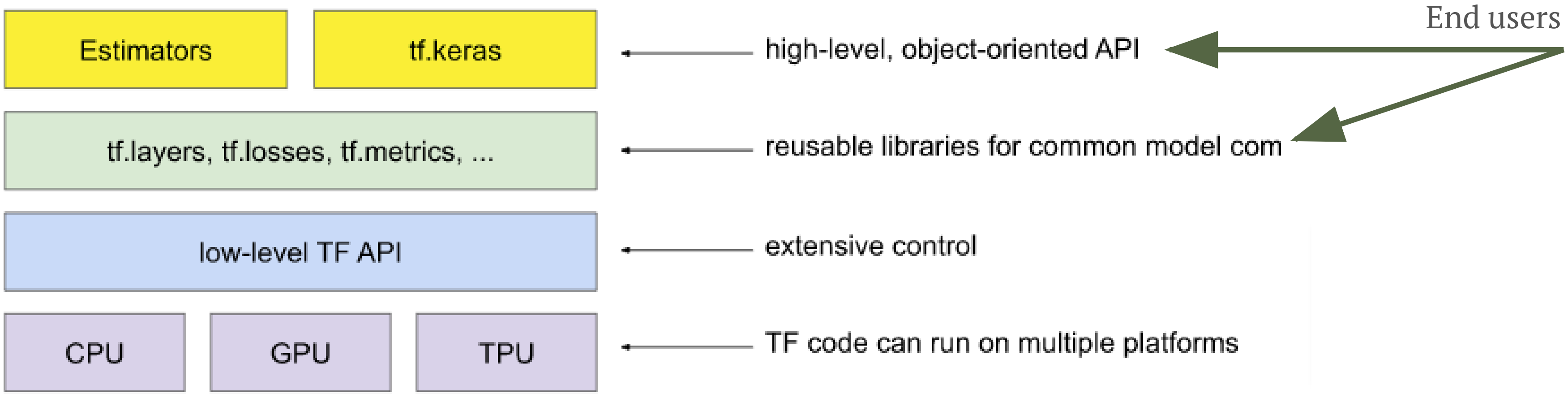
CPU

GPU

TPU

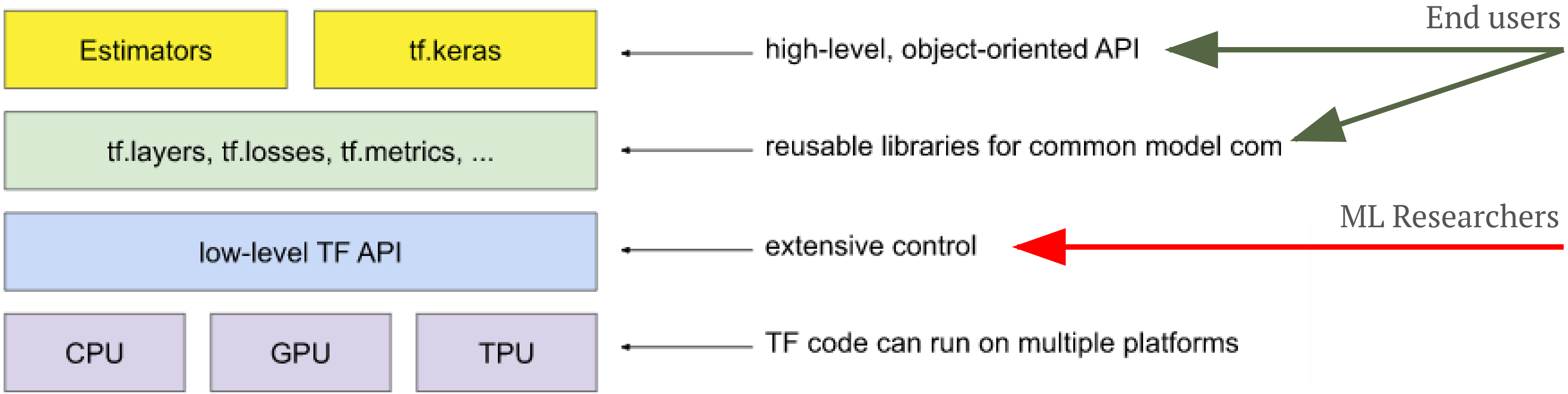
TF code can run on multiple platforms

Library structure

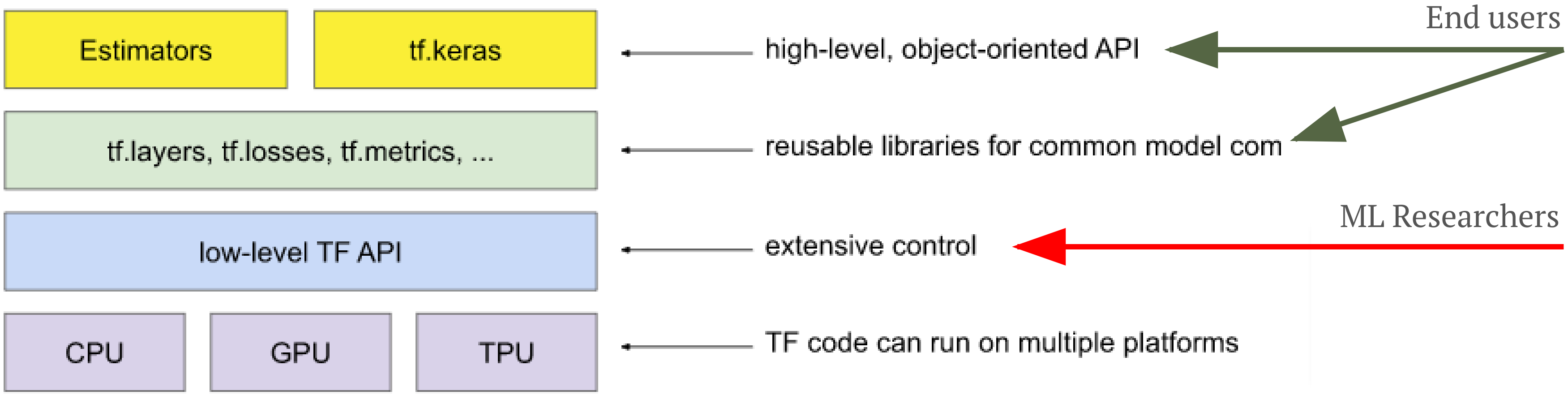


End users
Estimators
tf.keras
high-level, object-oriented API
tf.layers, tf.losses, tf.metrics, ...
reusable libraries for common model com
low-level TF API
extensive control
CPU
GPU
TPU
TF code can run on multiple platforms

Library structure


Estimators
tf.keras
tf.layers, tf.losses, tf.metrics, ...
low-level TF API
CPU
GPU
TPU
high-level, object-oriented API
reusable libraries for common model com
extensive control
TF code can run on multiple platforms
End users
ML Researchers

Library structure



End users

Estimators

tf.keras

high-level, object-oriented API

tf.layers, tf.losses, tf.metrics, ...

reusable libraries for common model com

ML Researchers

low-level TF API

extensive control

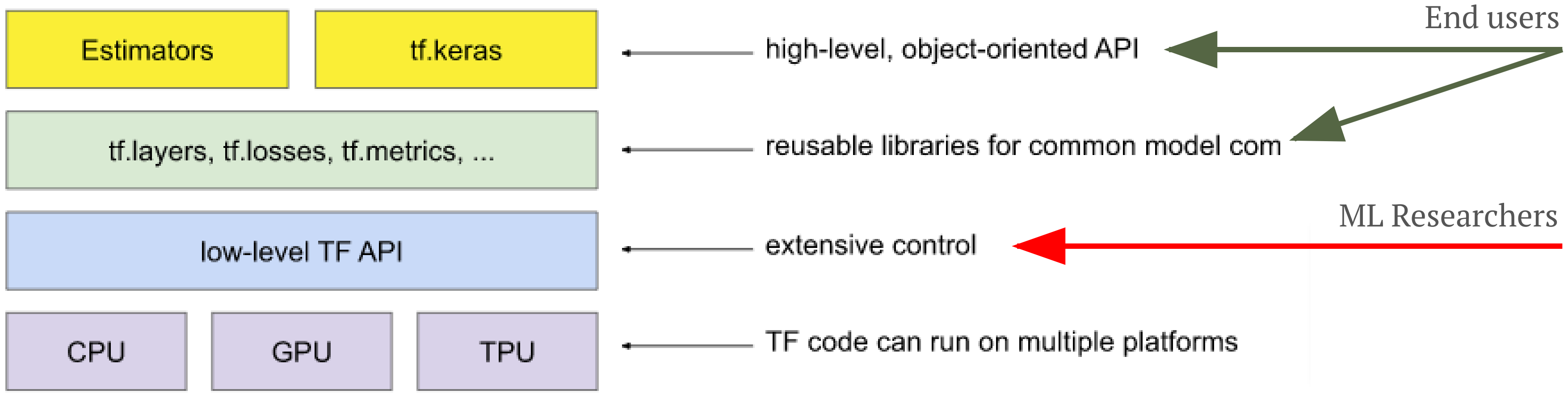
CPU

GPU

TPU

TF code can run on multiple platforms

Library structure



Estimators

tf.keras

tf.layers, tf.losses, tf.metrics, ...

low-level TF API

CPU

GPU

TPU

high-level, object-oriented API

reusable libraries for common model com

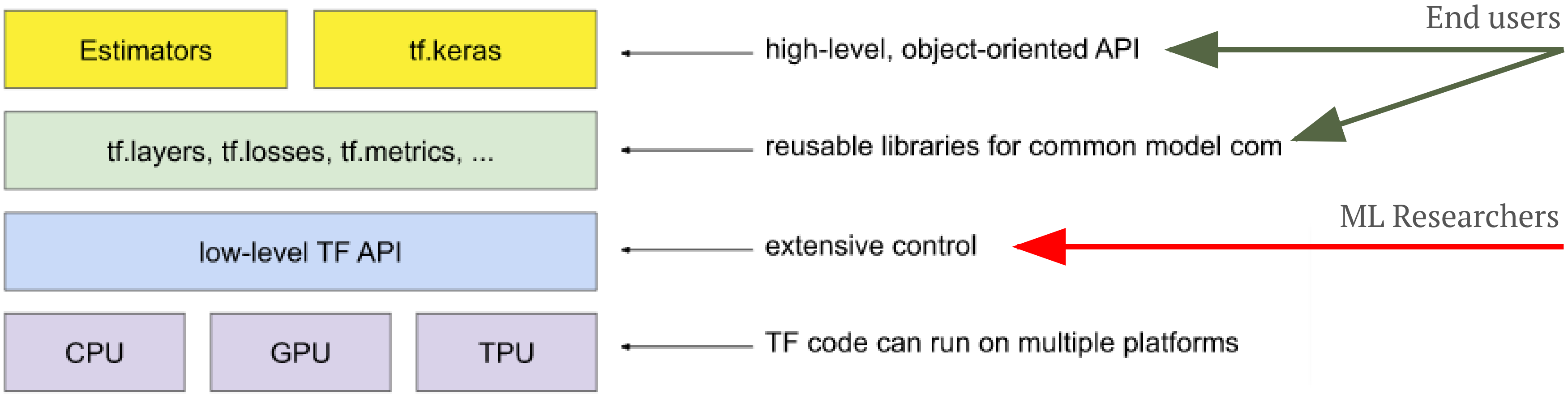
extensive control

TF code can run on multiple platforms

End users

ML Researchers

Library structure



Estimators

tf.keras

tf.layers, tf.losses, tf.metrics, ...

low-level TF API

CPU

GPU

TPU

high-level, object-oriented API

reusable libraries for common model com

extensive control

TF code can run on multiple platforms

End users

ML Researchers

Keras

Keras is a deep learning API writtent in Python, developed with focus on enabling fast experimentation.

- Simple – but not simplistic. Keras reduces developer cognitive load to free you to focus on the parts of the problem that really matter.
- Flexible – Keras adopts the principle of progressive disclosure of complexity: simple workflows should be quick and easy, while arbitrarily advanced workflows should be possible via a clear path that builds upon what you’ve already learned.
- Powerful – Keras provides industry-strength performance and scalability: it is used by organizations and companies including NASA, YouTube, or Waymo.

Tensorflow 2.X - Imports, Input and Output data

```python
import tensorflow as tf
import numpy as np

# Input data
x = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]], dtype=np.float32)

# Output data
y = np.array([[0],
              [1],
              [1],
              [0]], dtype=np.float32)
```

[1]: Numpy UltraQuick Tutorial

Tensorflow 2.X - Imports, Input and Output data

```
import tensorflow as tf
import numpy as np

# Input data
x = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]], dtype=np.float32)

# Output data
y = np.array([[0],
              [1],
              [1],
              [0]], dtype=np.float32)
```

[1]: Numpy UltraQuick Tutorial

Tensorflow 2.X - Imports, Input and Output data

```
import tensorflow as tf
import numpy as np

# Input data
x = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]], dtype=np.float32)

# Output data
y = np.array([[0],
              [1],
              [1],
              [0]], dtype=np.float32)
```

[1]: Numpy UltraQuick Tutorial

Tensorflow 2.X - Imports, Input and Output data

```
import tensorflow as tf
import numpy as np

# Input data
x = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]], dtype=np.float32)

# Output data
y = np.array([[0],
              [1],
              [1],
              [0]], dtype=np.float32)
```

[1]: Numpy UltraQuick Tutorial

Tensorflow 2.X - Imports, Input and Output data

```
import tensorflow as tf
import numpy as np

# Input data
x = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]], dtype=np.float32)

# Output data
y = np.array([[0],
              [1],
              [1],
              [0]], dtype=np.float32)
```

[1]: Numpy UltraQuick Tutorial

Tensorflow 2.X - Creating Model

Keras let us create models using Model or Sequential Class.

Sequential is the simplest model, with linear stack of layers, connectd sequentially 1-> 2 -> 3

Model is for more sophisticated use cases, for example connections across multiple layers 1 <-> 3

```
# Create simple Sequential Model
model = tf.keras.models.Sequential()
```

[1]: More in-depth difference between Model and Sequential

[2]: Example of advancel Model usage

Tensorflow 2.X - Layers, Activations

Keras provides some built-in layers and activation functions

```
tf.keras.activations.relu # Relu activation functions
tf.keras.activations.sigmoid # Sigmoid activation function
tf.keras.activations.tanh # Tanh activation function

tf.keras.layers.Dense # Regular, densely-connected NN layer
tf.keras.layers.Conv1d # 1D convolution layer
tf.keras.layers.dropout # Applies Dropout to the input, to prevent overfitting

# Example
tf.keras.layers.Dense(1, activation=tf.keras.activations.sigmoid) # 1 Neuron layer with sigmoid activation function
```

Tensorflow 2.X - Compile, train and predict

First, we need to configure the model for training [1]

```
model.compile(
    optimizer='Adam', # optimizer name or instance
    loss="mse", # Mean squared error
    metrics=["mae", "acc"] # Loss function - Mean Absolute Error, Accuracy
)
```

Learning [2]

```
model.fit(
    x=None, # Input data
    y=None, # Target data
    batch_size=None, # Number of samples per gradient update
    epochs=1, # Number of epochs to train the model.
    shuffle=True, # Whether to shuffle the training data before each epoch
)
```

---

1. All compile options ↩

2. More about fitting ↩

XOR example

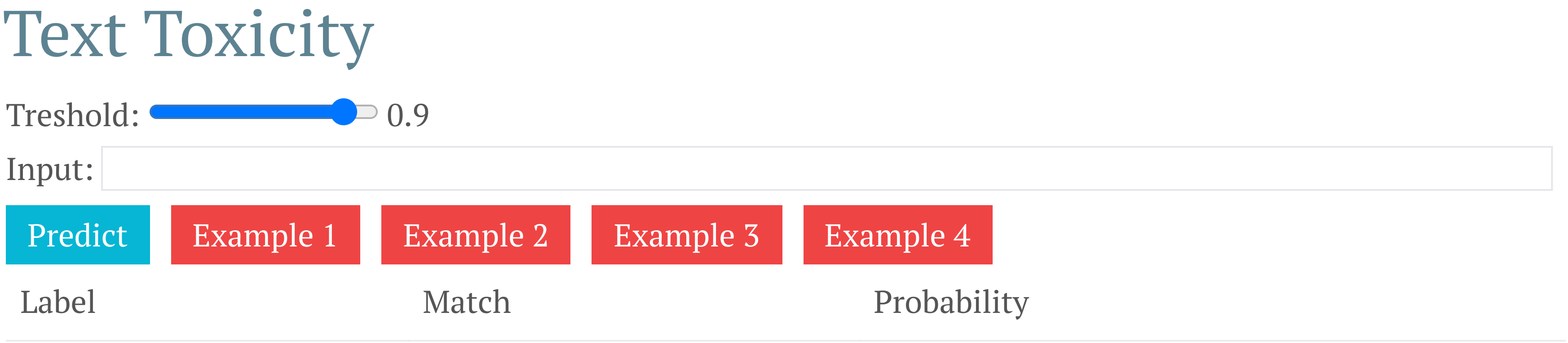
XOR TF implementation - building model

```
import tensorflow as tf
import numpy as np

# Examples
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
# Labels
y = np.array([[0], [1], [1], [0]], dtype=np.float32)

model = tf.keras.models.Sequential()
model.add(tf.keras.Input(shape=(2,)))
model.add(tf.keras.layers.Dense(2,
  activation=tf.keras.activations.sigmoid,
  kernel_initializer=tf.initializers.Constant(0.5)))
model.add(tf.keras.layers.Dense(1, activation=tf.keras.activations.sigmoid))

# Compile the model
model.compile(
  optimizer=tf.keras.optimizers.Adam(learning_rate=0.1),
  loss=tf.keras.losses.MeanSquaredError(),
  metrics=['mse', 'binary_accuracy'])

model.fit(x, y, batch_size=1, epochs=500)
```

XOR TF Implementation - prediction

```
predictions = model.predict(x)
print(predictions)

#[[0.01753041]
# [0.9767829 ]
# [0.97674406]
# [0.04294848]]
```

TensorFlow.js



Text Toxicity

Treshold: 0.9

Input:

Label	Match	Probability

Text Toxicity implementation - whole component

```
import '@tensorflow/tfjs';
import { load } from '@tensorflow-models/toxicity'
import type { ToxicityClassifier } from '@tensorflow-models/toxicity';
import { ref, watch } from 'vue';
import type { Ref } from 'vue';
import { debounce } from 'lodash';
import { ClassificationResult } from './models'

const examples = [...] // small censorship 🚫
const treshold: Ref<number> = ref(0.9)
let model: ToxicityClassifier = await load(treshold.value, []);
const input: Ref<string> = ref('');
const result: Ref<ClassificationResult[]> = ref([]);
watch(treshold, debounce(loadModel, 500))

async function loadModel(treshold: number) {
  model = await load(treshold, [])
}

async function predict() {
  result.value = [];
  result.value = await model.classify([input.value]);
}
```

Text Toxicity implementation - whole component

```
import '@tensorflow/tfjs';
import { load } from '@tensorflow-models/toxicity'
import type { ToxicityClassifier } from '@tensorflow-models/toxicity';
import { ref, watch } from 'vue';
import type { Ref } from 'vue';
import { debounce } from 'lodash';
import { ClassificationResult } from './models'

const examples = [...] // small censorship 🚫
const treshold: Ref<number> = ref(0.9)
let model: ToxicityClassifier = await load(treshold.value, []);
const input: Ref<string> = ref('');
const result: Ref<ClassificationResult[]> = ref([]);
watch(treshold, debounce(loadModel, 500))

async function loadModel(treshold: number) {
  model = await load(treshold, [])
}

async function predict() {
  result.value = [];
  result.value = await model.classify([input.value]);
}
```

Learn More

Documentations · GitHub · Showcases

Sources

- TensorFlow Docs
- TensorFlow Guides
- Keras Docs
- Google Developers - Introduction to TensorFlow
- TF Model Class usage
- Stackoverflow
- XOR
- Toxicity classifier