TensorBoard

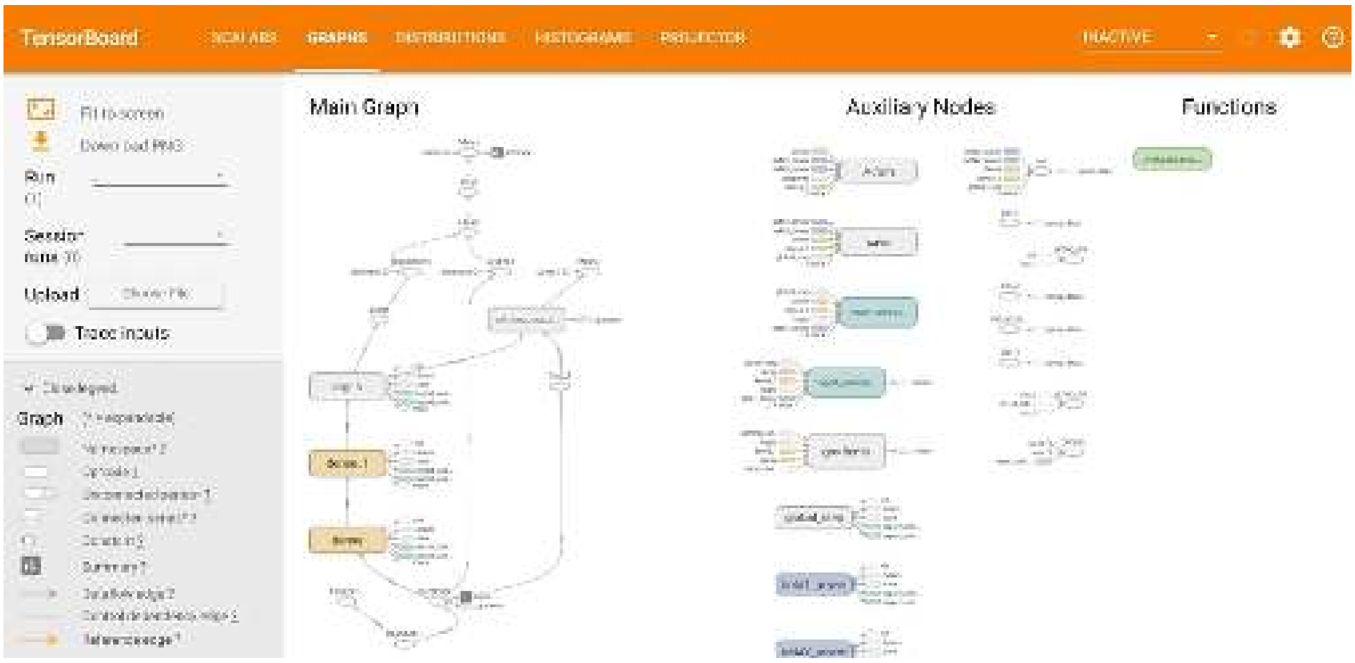
Discover how TensorBoard can be used to visualize machine learning.

Chapter Goals:

- Learn about TensorBoard and how to track the progression of training values
- Specify training values to be shown in TensorBoard

A. Training visualizations

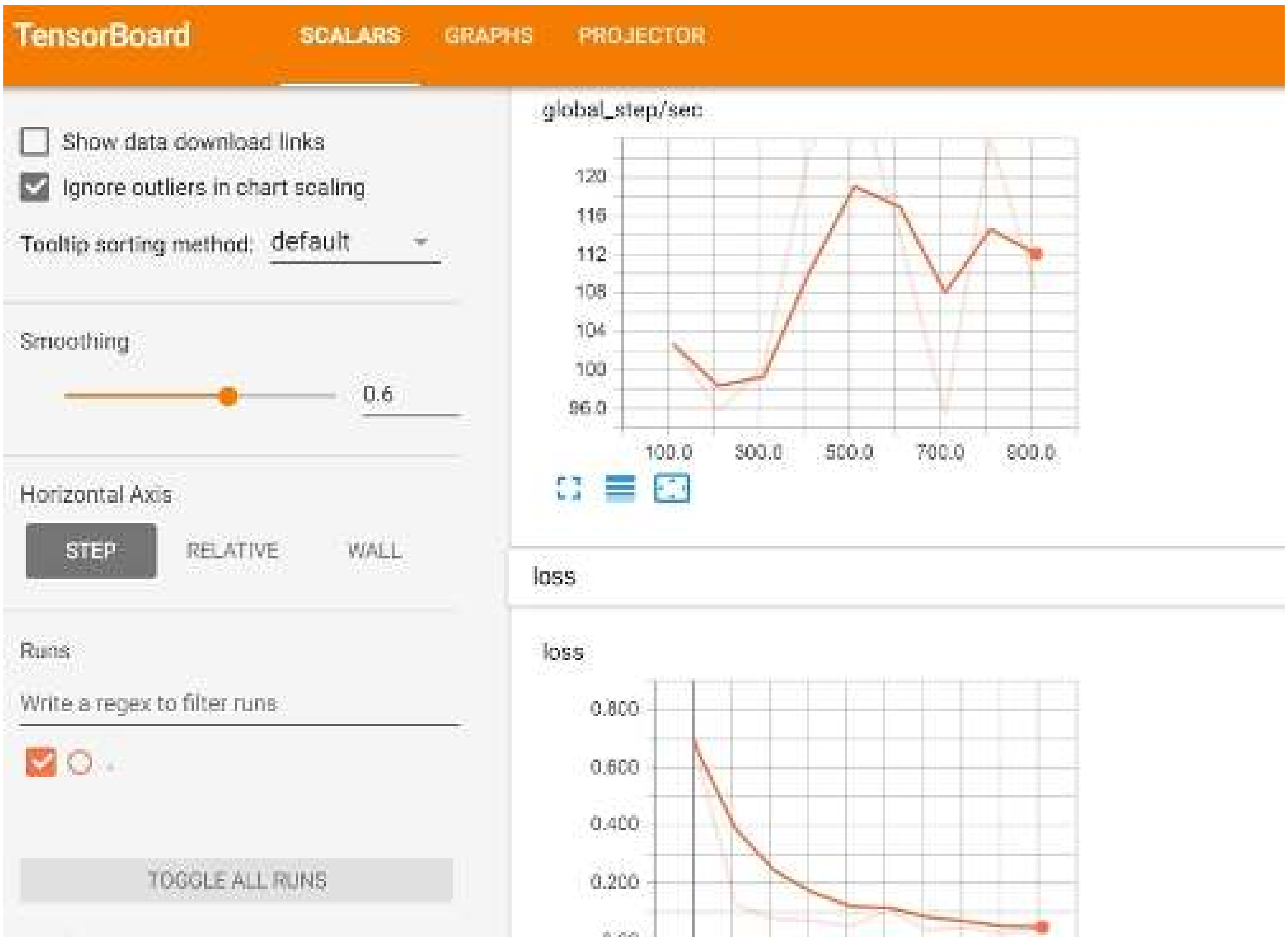
When training a complex neural network, it is useful to have visualizations of the compuation graph and important values to make sure everything is correct. In TensorFlow, there is a tool known as TensorBoard, which lets us visualize all the important aspects of a model.



Computation graph structure of a neural network shown in TensorBoard.

TensorBoard works by reading in an *events file*, which contains all the model data we want visualized. When training a model, the events file is stored in the same directory as the model *checkpoint* (which we'll discuss in the next chapter). The events file automatically contains the computation graph structure, as well as the loss and training speed (in iterations per second).

structure, as well as the loss and training speed (in iterations per second).



Show data download links

Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP

Default events file values shown in TensorBoard. Note that "global_step/sec" represents the training iterations per second.

You can run TensorBoard from the command line with the built-in **tensorboard** module (which comes as part of the TensorFlow library). You just need to specify the directory containing the events file. TensorBoard will then be running in the browser at *http://localhost:6006*.

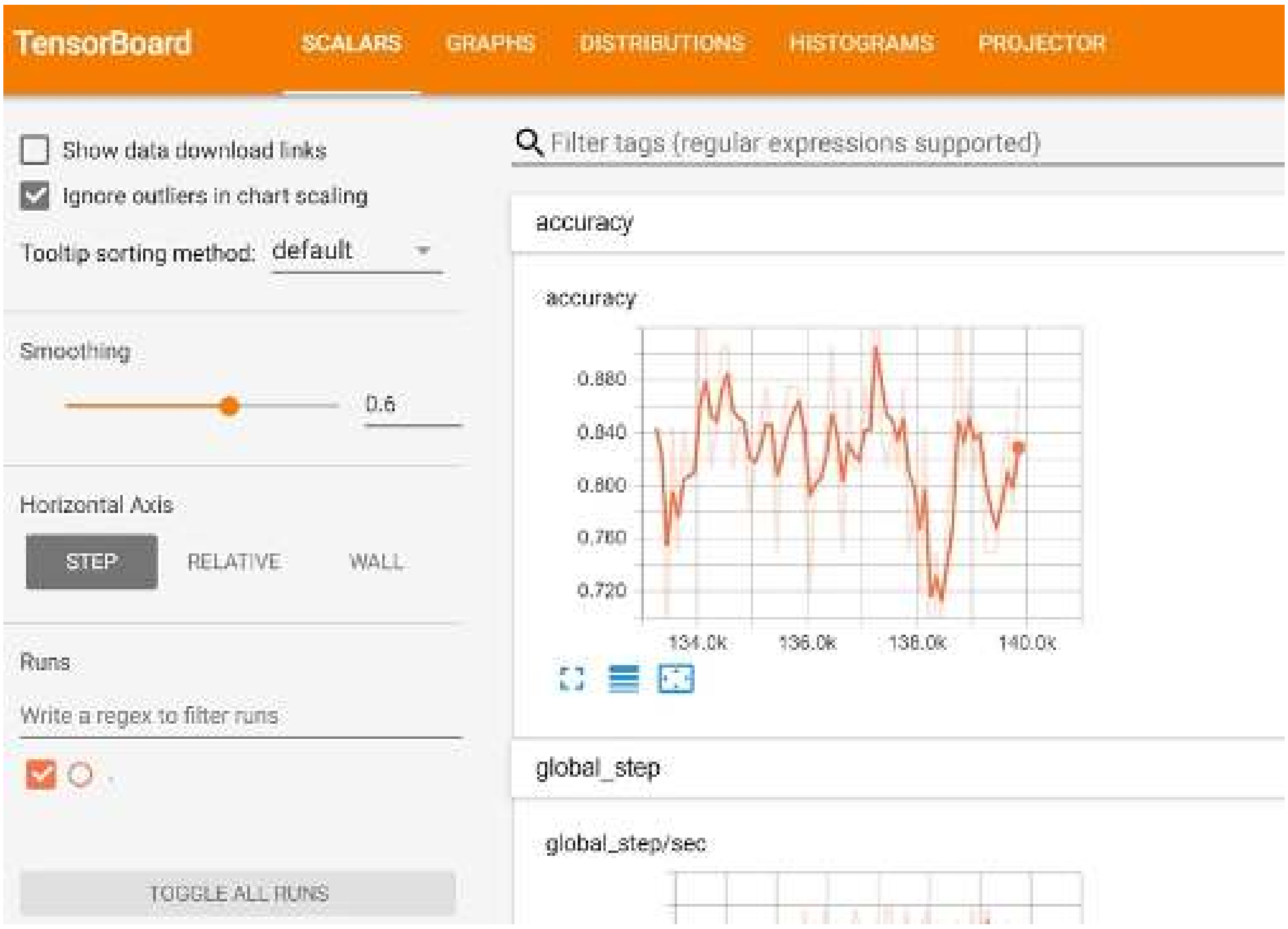
```
tensorboard --logdir=/home/ckpt_dir
```

This command line statement runs TensorBoard. The events file is located in the /home/ckpt_dir directory.

B. Tracking values

Apart from the default events file values, we can also specify custom values to track in TensorBoard. To do this, we just need to call `tf.summary.scalar` in our code.

The function takes in two required arguments. The first argument is the label name for the visualization in TensorFlow. The second argument is the scalar (i.e. single numeric value) tensor that will be visualized. The tensor's values will be plotted with respect to the training iterations.

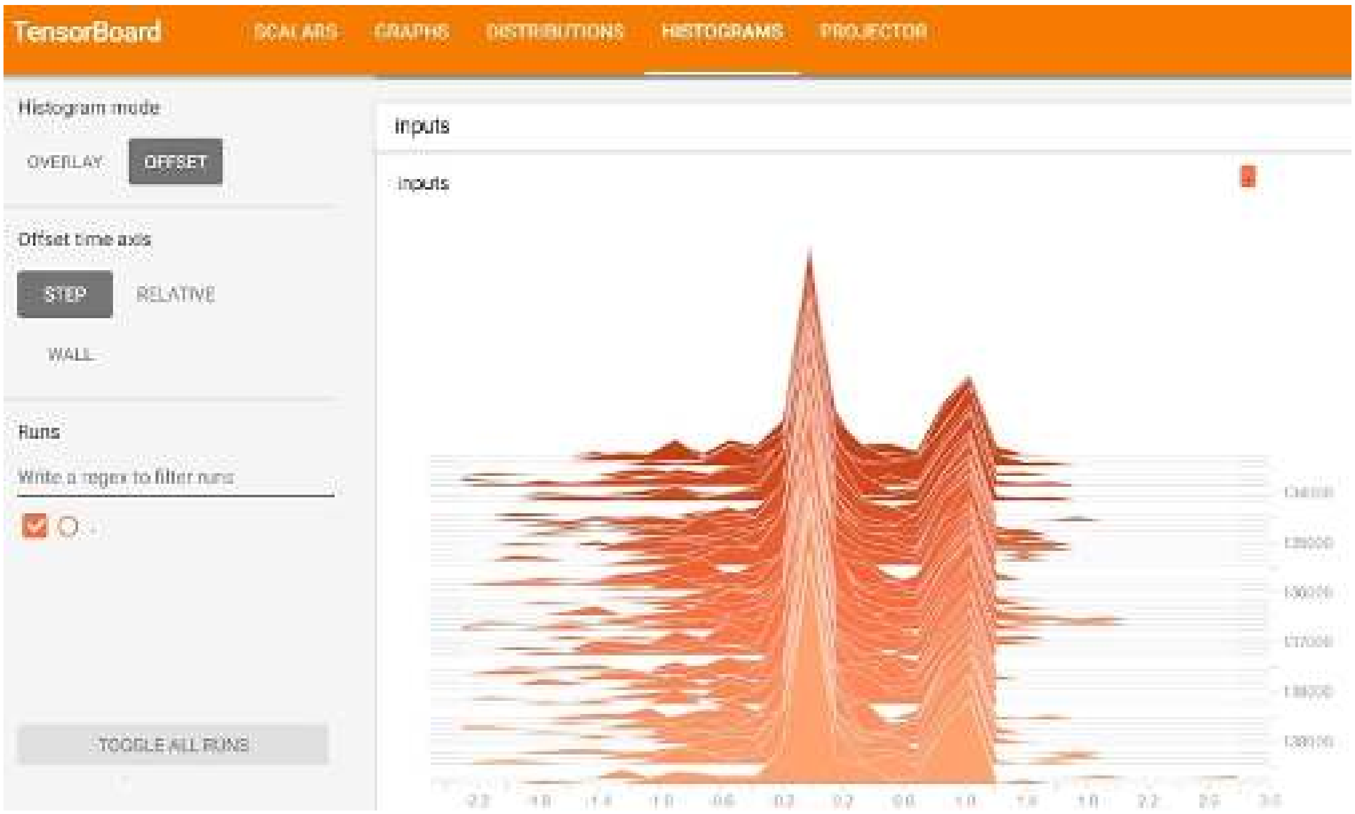


☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis [✓ STEP] RELATIVE WALL

Visualizing the model accuracy in TensorBoard. The plot shows the accuracy on the y-axis and training iteration on the x-axis.

We can also visualize the distribution of the values for a particular layer in the model. For example, we can view the distribution of the data in the input layer, or we can view the distribution of the weights in a particular hidden layer.

The function we use to visualize a distribution is `tf.summary.histogram`. This function takes in the same arguments as `tf.summary.scalar`.



Histogram mode
OVERLAY [✓ OFFSET]

Offset time axis
[✓ STEP] RELATIVE WALL

Visualizing the distribution of the input layer in TensorBoard. The histogram shows the frequency of value appearances, with higher peaks representing more common values.

Time to Code!

The first seven chapters of this section of the course deals with creating a classification model in TensorFlow, represented by the `ClassificationModel` object. The function that you'll be working on this chapter and the next is `run_model_training`. This function will run training for a classification MLP model and log results to TensorBoard.

The function calls two helpers:

- `dataset_from_numpy`: creates a dataset from NumPy data
- `run_model_setup`: sets up the MLP model

Both functions are shown below.

```
import numpy as np
import tensorflow as tf

class ClassificationModel(object):
    def __init__(self, output_size):
        self.output_size = output_size

    # See the "Efficient Data Processing Techniques" section for details
    def dataset_from_numpy(self, input_data, batch_size, labels=None, is_training=True, num_e
        dataset_input = input_data if labels is None else (input_data, labels)
        dataset = tf.data.Dataset.from_tensor_slices(dataset_input)
        if is_training:

            dataset = dataset.shuffle(len(input_data)).repeat(num_epochs)
        return dataset.batch(batch_size)

    # See the "Machine Learning for Software Engineers" course on Educative
    def run_model_setup(self, inputs, labels, hidden_layers, is_training, calculate_accuracy=
        layer = inputs
        for num_nodes in hidden_layers:
            layer = tf.layers.dense(layer, num_nodes,
                activation=tf.nn.relu)
        logits = tf.layers.dense(layer, self.output_size,
            name='logits')
        self.probs = tf.nn.softmax(logits, name='probs')
        self.predictions = tf.argmax(
            self.probs, axis=-1, name='predictions')
        if calculate_accuracy:
            class_labels = tf.argmax(labels, axis=-1)
            is_correct = tf.equal(
                self.predictions, class_labels)
            is_correct_float = tf.cast(
                is_correct,
                tf.float32)
            self.accuracy = tf.reduce_mean(
                is_correct_float)
        if labels is not None:
            labels_float = tf.cast(
                labels, tf.float32)
            cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(
                labels=labels_float,
                logits=logits)
            self.loss = tf.reduce_mean(
                cross_entropy)
        if is_training:
            adam = tf.train.AdamOptimizer()
            self.train_op = adam.minimize(
                self.loss, global_step=self.global_step)
```

In this chapter, you'll be creating another helper function called `add_to_tensorboard`, which adds metrics to log in TensorBoardg.

It's useful to keep track of how the model accuracy changes while training. Therefore, we'll want to make sure it's plotted in TensorBoard.

Call `tf.summary.scalar` with `'accuracy'` as the first argument and `self.accuracy` as the second argument.

We also want to store the distribution of the input layer (represented by `inputs`) in our TensorBoard. To visualize the distribution, we'll make sure to store it as a histogram.

Call `tf.summary.histogram` with `'inputs'` as the first argument and `inputs` as the second argument.

as the second argument.

```
import numpy as np
import tensorflow as tf

class ClassificationModel(object):
    def __init__(self, output_size):
        self.output_size = output_size

    # Adds metrics to TensorBoard
    def add_to_tensorboard(self, inputs):
        # CODE HERE
        pass

    # See the "Efficient Data Processing Techniques" section for details
    def dataset_from_numpy(self, input_data, batch_size, labels=None, is_training=True, num_e
        dataset_input = input_data if labels is None else (input_data, labels)
        dataset = tf.data.Dataset.from_tensor_slices(dataset_input)
        if is_training:
            dataset = dataset.shuffle(len(input_data)).repeat(num_epochs)
        return dataset.batch(batch_size)

    # See the "Machine Learning for Software Engineers" course on Educative
    def run_model_setup(self, inputs, labels, hidden_layers, is_training, calculate_accuracy=
        layer = inputs
        for num_nodes in hidden_layers:
            layer = tf.layers.dense(layer, num_nodes,
                activation=tf.nn.relu)
        logits = tf.layers.dense(layer, self.output_size,
            name='logits')
        self.probs = tf.nn.softmax(logits, name='probs')
        self.predictions = tf.argmax(
            self.probs, axis=-1, name='predictions')
        if calculate_accuracy:
            class_labels = tf.argmax(labels, axis=-1)
            is_correct = tf.equal(
                self.predictions, class_labels)
            is_correct_float = tf.cast(
                is_correct,
                tf.float32)
            self.accuracy = tf.reduce_mean(
                is_correct_float)
        if labels is not None:
            labels_float = tf.cast(
                labels, tf.float32)
            cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(
                labels=labels_float,
                logits=logits)
            self.loss = tf.reduce_mean(
                cross_entropy)
        if is_training:
            adam = tf.train.AdamOptimizer()
            self.train_op = adam.minimize(
                self.loss, global_step=self.global_step)

    # Run training of the classification model
    def run_model_training(self, input_data, labels, hidden_layers, batch_size, num_epochs, c
        self.global_step = tf.train.get_or_create_global_step()
        dataset = self.dataset_from_numpy(input_data, batch_size,
            labels=labels, num_epochs=num_epochs)
        iterator = dataset.make_one_shot_iterator()
        inputs, labels = iterator.get_next()


        self.run_model_setup(inputs, labels, hidden_layers, True)
        self.add_to_tensorboard(inputs)
```