Intro

- In this video we will train our first model in TensorFlow

Optimizers in TensorFlow

- Let's define **f** as a square of variable **x**:

```
import numpy as np
import tensorflow as tf

tf.reset_default_graph()
x = tf.get_variable("x", shape=(), dtype=tf.float32)
f = x ** 2
```

- Let's say we want to *minimize* the value of **f** w.r.t **x**:

```
optimizer = tf.train.GradientDescentOptimizer(0.1)
step = optimizer.minimize(f, var_list=[x])
```

Trainable variables

- You don’t have to specify all the optimized variables:

```
step = optimizer.minimize(f, var_list=[x])
step = optimizer.minimize(f)
```

- Because all variables are **trainable** by default**:**

```
x = tf.get_variable("x", shape=(), dtype=tf.float32)
x = tf.get_variable("x", shape=(), dtype=tf.float32, trainable=True)
```

- You can get all of them:

```
tf.trainable_variables()
```

- Output:

```
[<tf.Variable 'x:0' shape=() dtype=float32_ref>]
```

Making gradient descent steps

- Now we need to create a **session** and **initialize** variables:

```
s = tf.InteractiveSession()
s.run(tf.global_variables_initializer())
```

- We are ready to make 10 **gradient descent** steps:

```
for i in range(10):
    _, curr_x, curr_f = s.run([step, x, f])
    print(curr_x, curr_f)
```

- Output:

```
0.448929 0.314901
0.359143 0.201537
…
0.0753177 0.00886368
0.0602542 0.00567276
```

GD step is already applied to x

Logging with tf.Print

- We can **evaluate** tensors and print them like this:

```
for i in range(10):
    _, curr_x, curr_f = s.run([step, x, f])
    print(curr_x, curr_f)
```

- Or we can pass our tensor of interest through **tf.Print**:

```
...
f = x ** 2
f = tf.Print(f, [x, f], "x, f:")
...
for i in range(10):
    s.run([step, f])
```

```
x, f:[1.5879565][2.521606]
x, f:[1.2703652][1.6138278]
...
x, f:[0.26641488][0.070976891]
x, f:[0.2131319][0.04542521]
```

This is Jupyter Server stdout.
Not visible in the Notebook!

Logging with TensorBoard

- We can add so-called **summaries**:

```
tf.summary.scalar('curr_x', x)
tf.summary.scalar('curr_f', f)
summaries = tf.summary.merge_all()
```

- This is how we log these summaries:

Run number

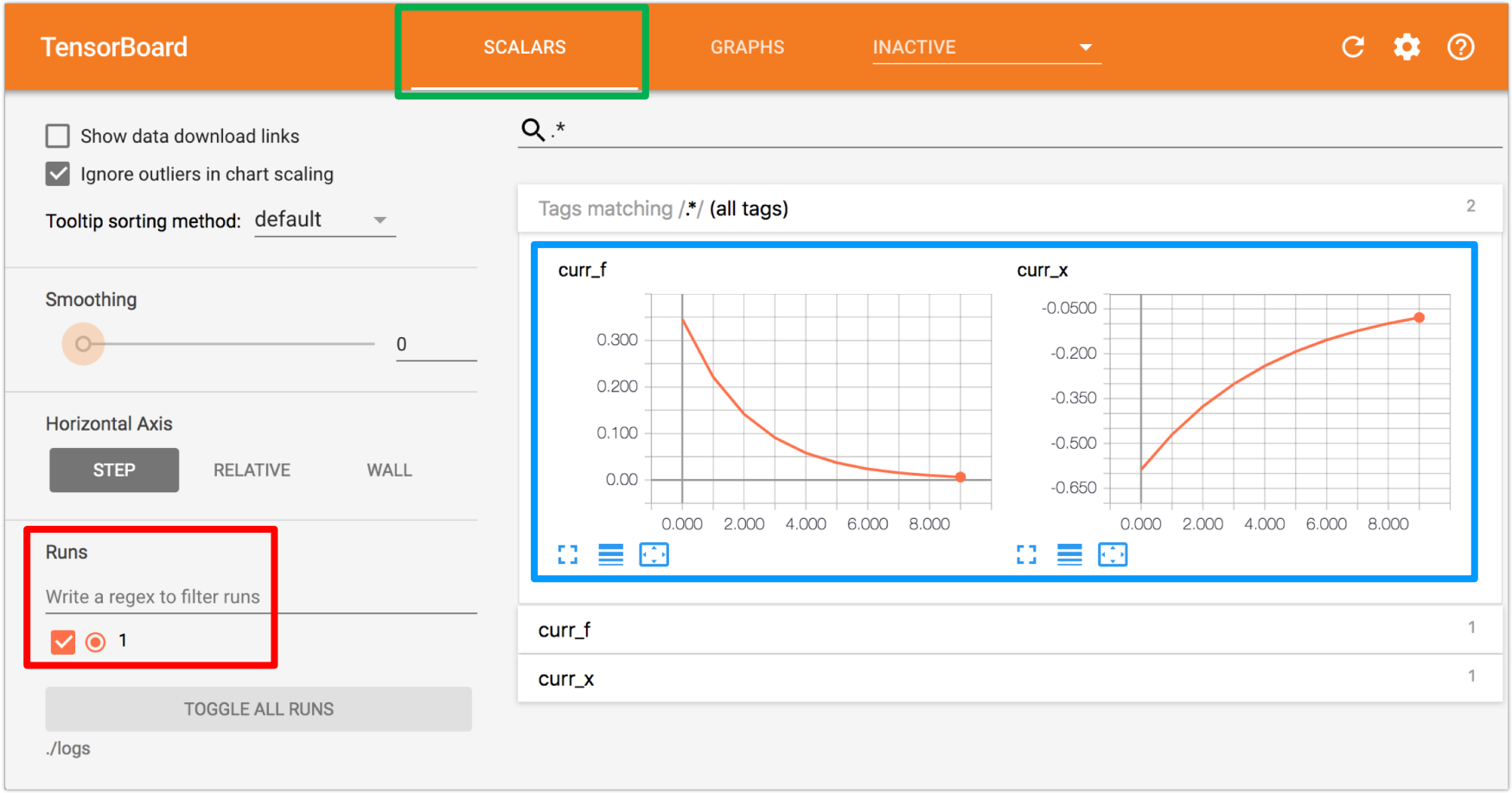
```
s = tf.InteractiveSession()
summary_writer = tf.summary.FileWriter("logs/1", s.graph)
s.run(tf.global_variables_initializer())
for i in range(10):
    _, curr_summaries = s.run([step, summaries])
    summary_writer.add_summary(curr_summaries, i)
    summary_writer.flush()
```

Launching TensorBoard

- Now you can launch TensorBoard via bash:

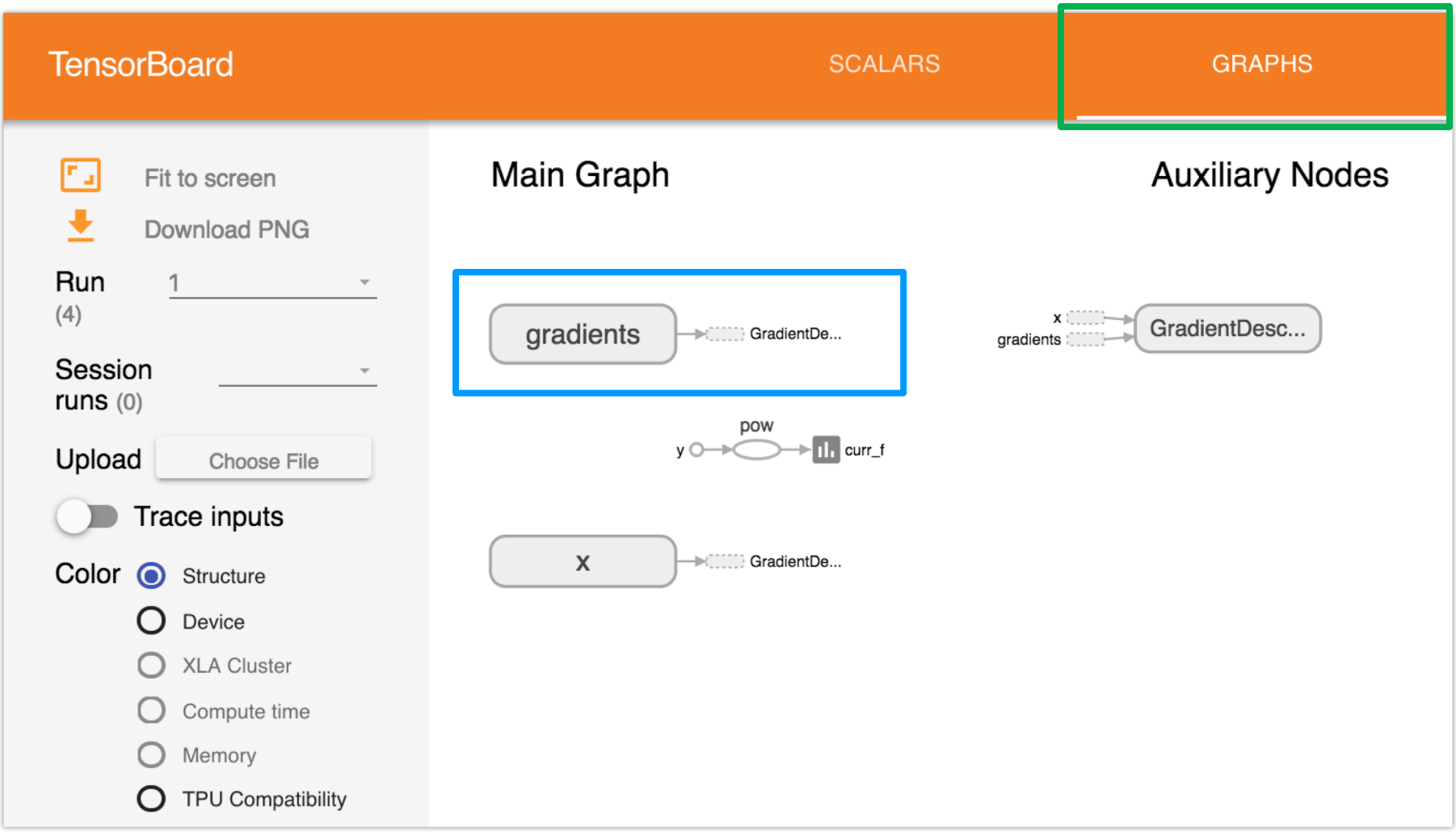
  tensorboard --logdir=./logs

- And open **http://127.0.0.1:6006** in your browser.



Visualizing graph in TensorBoard

- You can see that **gradients computation** is a part of our graph:



Solving a linear regression

- Let's generate a model dataset:

```
N = 1000
D = 3
x = np.random.random((N, D))
w = np.random.random((D, 1))
y = x @ w + np.random.randn(N, 1) * 0.20
```

Solving a linear regression

- We will need **placeholders** for input data:

```
tf.reset_default_graph()
features = tf.placeholder(tf.float32, shape=(None, D))
target = tf.placeholder(tf.float32, shape=(None, 1))
```

- This is how we make **predictions**:

```
weights = tf.get_variable("w", shape=(D, 1), dtype=tf.float32)
predictions = features @ weights
```

- And define our **loss**:

```
loss = tf.reduce_mean((target - predictions) ** 2)
```

- And **optimizer**:

```
optimizer = tf.train.GradientDescentOptimizer(0.1)
step = optimizer.minimize(loss)
```

Solving a linear regression

- Gradient descent:

```
s = tf.InteractiveSession()
s.run(tf.global_variables_initializer())
for i in range(300):
    _, curr_loss, curr_weights = s.run(
        [step, loss, weights], feed_dict={features: x, target: y})
    if i % 50 == 0:
        print(curr_loss)
```

Filling placeholders

- Ground truth weights:

[0.11649134,0.82753164,0.46924019]

- Found weights:

[0.13715988,0.79555332,0.47024861]

Model checkpoints

- We can save variables' state with **tf.train.Saver**:

```
s = tf.InteractiveSession()
saver = tf.train.Saver(tf.trainable_variables())
s.run(tf.global_variables_initializer())
for i in range(300):
    _, curr_loss, curr_weights = s.run(
        [step, loss, weights], feed_dict={features: x, target: y})
    if i % 50 == 0:
        saver.save(s, "logs/2/model.ckpt", global_step=i)
        print(curr_loss)
```

Model checkpoints

- We can **list** last checkpoints:

```
saver.last_checkpoints
```

```
['logs/2/model.ckpt-50', 'logs/2/model.ckpt-100',
'logs/2/model.ckpt-150', 'logs/2/model.ckpt-200',
'logs/2/model.ckpt-250']
```

- We can **restore** a previous checkpoint like this:

```
saver.restore(s, "logs/2/model.ckpt-50")
```

- Only variables' values are restored, which means that you need to define a graph in *the same* way **before restoring** a checkpoint.

Summary

- TensorFlow has **built-in optimizers** that do back-propagation automatically.

- TensorBoard provides **tools for visualizing** your training progress.

- TensorFlow allows you to **checkpoint** your graph to restore its state later *(you need to define it in exactly the same way though)*