

第四部分

实战 TensorFlow 房价预测



扫描二维码

试看/购买 《TensorFlow 快速入门与实战》 视频课程

第四部分 目录

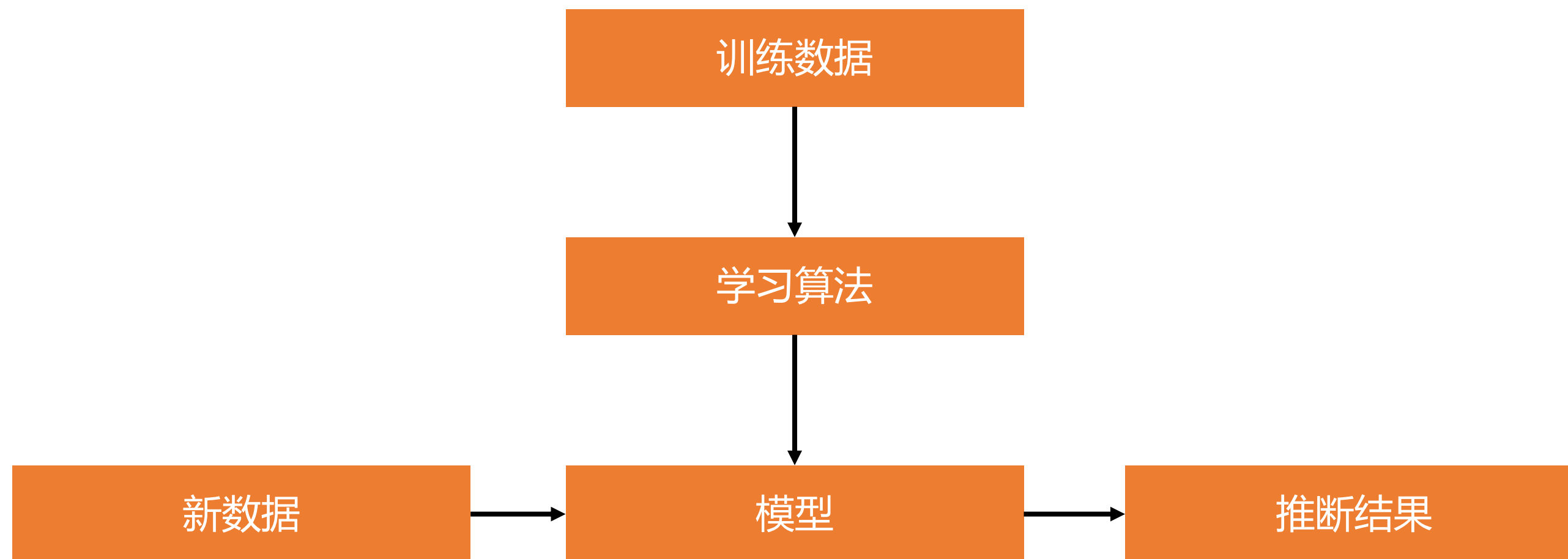
- 房价预测模型介绍
- 使用 TensorFlow 实现房价预测模型
- 使用 TensorBoard 可视化模型数据流图
- 实战 TensorFlow 房价预测

房价预测模型介绍

前置知识：监督学习（Supervised Learning）

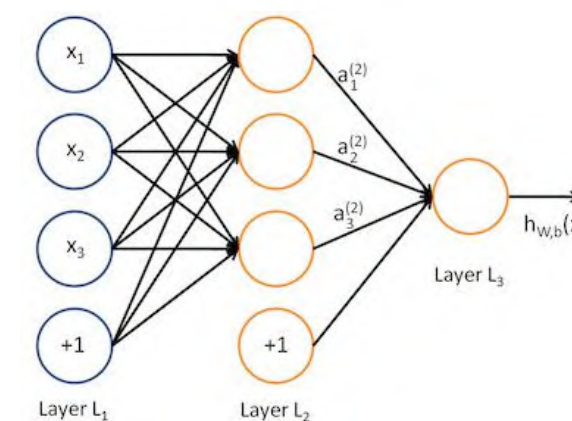
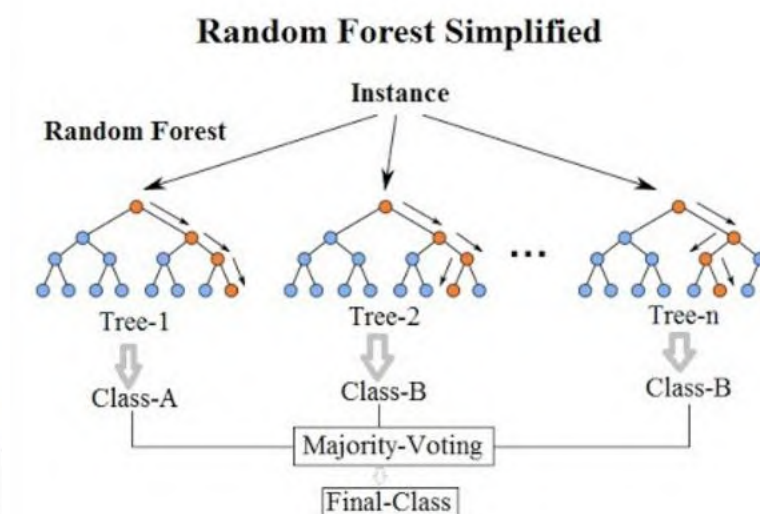
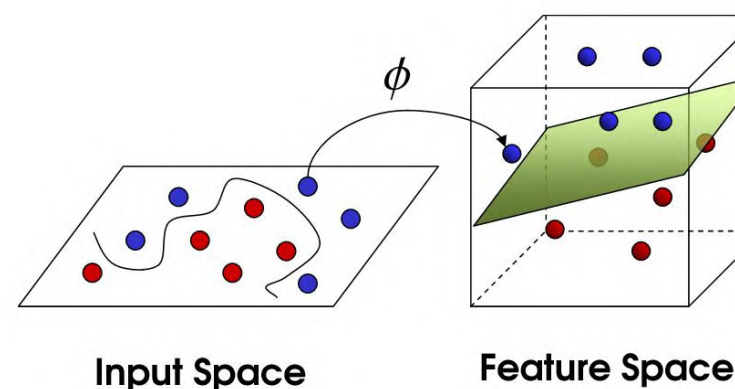
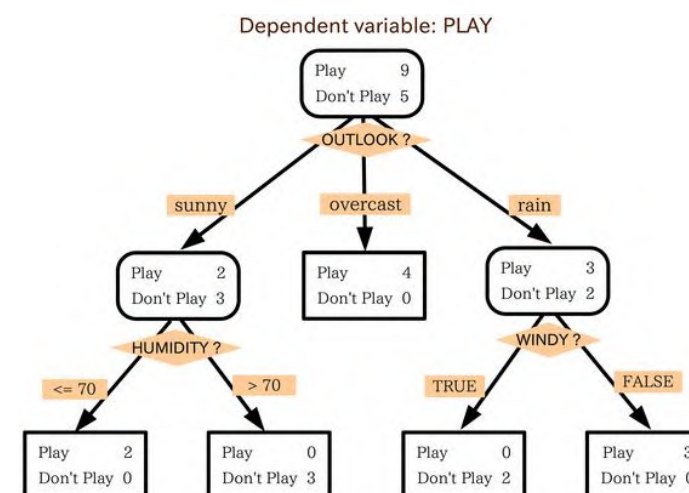
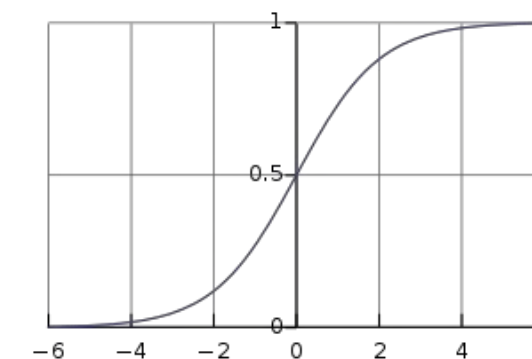
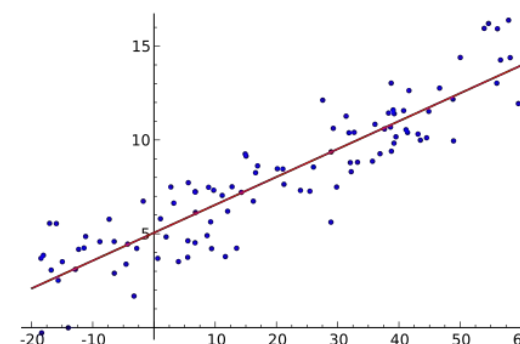
监督学习是机器学习的一种方法，指从训练数据（输入和预期输出）中学到一个模型（函数），并根据模型可以推断新实例的方法。

函数的输出通常为一个连续值（回归分析）或类别标签（分类）。



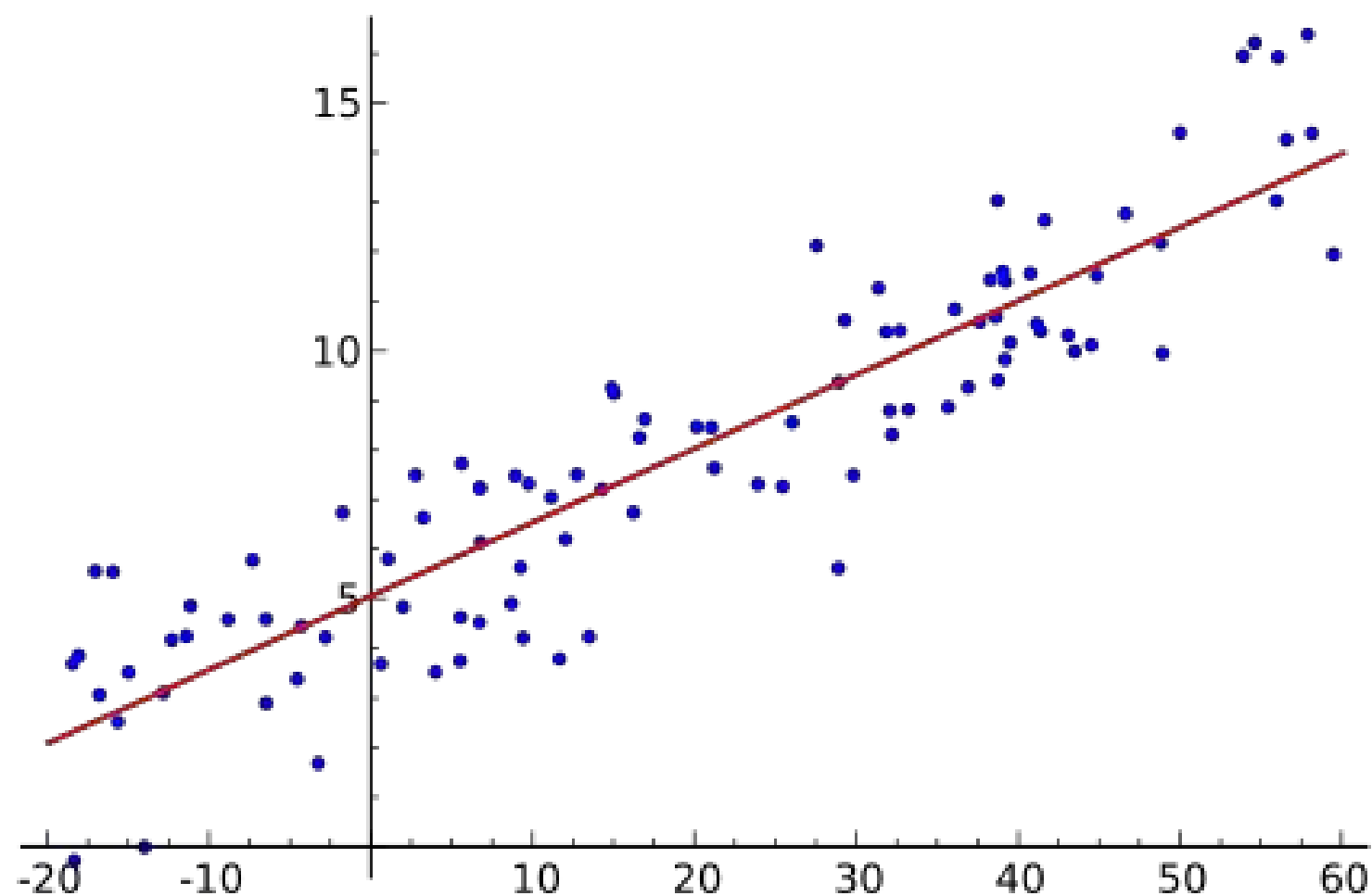
前置知识：监督学习典型算法

- 线性回归（Linear Regression）
- 逻辑回归（Logistic Regression）
- 决策树（Decision Tree）
- 随机森林（Random Forest）
- 最近邻算法（k-NN）
- 朴素贝叶斯（Naive Bayes）
- 支持向量机（SVM）
- 感知器（Perceptron）
- 深度神经网络（DNN）



前置知识：线性回归

在统计学中，线性回归是利用称为线性回归方程的**最小二乘函数**对**一个或多个**自变量和因变量之间关系进行建模的一种回归分析。这种函数是一个或多个称为回归系数的模型参数的线性组合。



前置知识：单变量线性回归

理想函数

$$y = b + wx$$

↑ ↑

假设函数

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 = \theta^T x$$

$$h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 = \theta^T x \quad (x_0=1)$$

损失值（误差）

$$loss = y - h_{\theta}(x)$$

前置知识：单变量线性回归

为了从一组样本 $(x^{(i)}, y^{(i)})$ (其中 $i = 1, 2, \dots, n$) 之中估计最合适 (误差最小) 的 θ_0 和 θ_1 , 通常采用最小二乘法, 其优化目标为最小化残差平方和:

$$J(\theta) = \frac{1}{n} \sum_i^n (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

为了消除求导时产生的系数, 将目标函数设置为 $\frac{1}{2}J(\theta)$, 不影响优化策略与梯度下降方向。即:

$$J(\theta) = \frac{1}{2n} \sum_i^n (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

前置知识：梯度下降

模型参数 θ_0 和 θ_1 在优化目标函数的每轮迭代中，按如下表达式更新：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

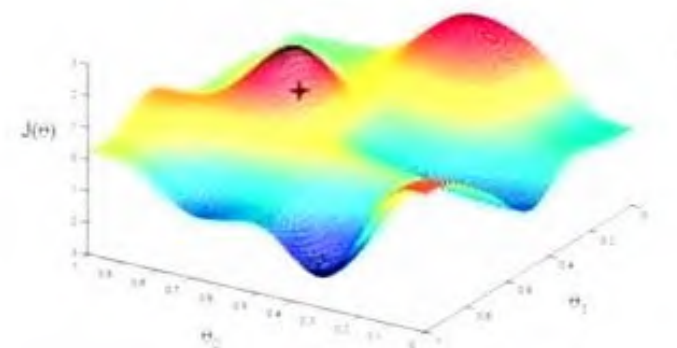
代入 $J(\theta)$ 求导，结果如下：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_i^n (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

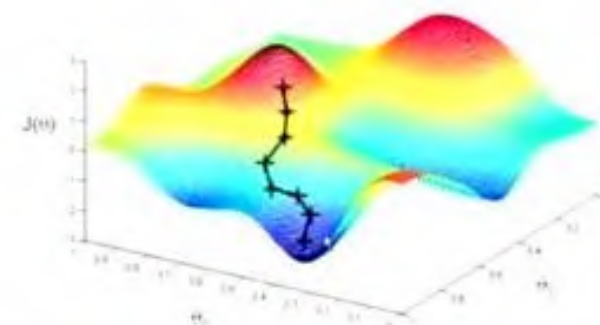
$$\theta_j := \theta_j - \alpha \frac{1}{2n} \sum_i^n 2(h_{\theta}(x^{(i)}) - y^{(i)}) (x_j^{(i)})$$

$$\theta_j := \theta_j - \alpha \frac{1}{n} \sum_i^n (h_{\theta}(x^{(i)}) - y^{(i)}) (x_j^{(i)})$$

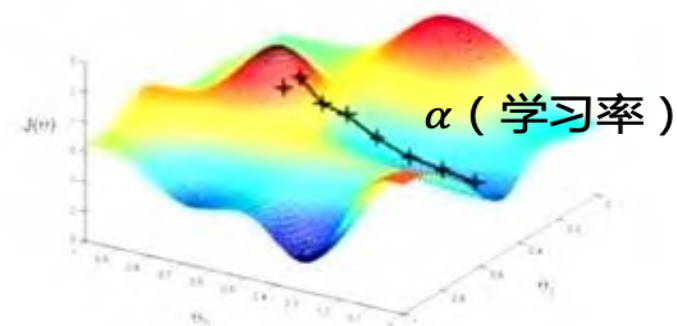
Gradient Descent



Gradient Descent



Gradient Descent



前置知识：多变量线性回归

理想函数

$$y = w_0 + w_1x_1 + w_2x_2$$

$$\vec{y} = W^T X$$

假设函数

$$h_{\theta}(X) = \theta^T X = \theta_0 + \theta_1x_1 + \theta_2x_2$$

损失值（误差）

$$loss = \vec{y} - h_{\theta}(X)$$

前置知识：梯度下降

为了从一组样本 $(x_1^{(i)}, x_2^{(i)}, y^{(i)})$ (其中 $i = 1, 2, \dots, n$) 之中估计最合适 (误差最小) 的模型参数 θ_j (θ_0, θ_1 和 θ_2) , 目标函数如下 :

$$J(\theta) = \frac{1}{2n} (h_{\theta}(X) - \vec{y})^T (h_{\theta}(X) - \vec{y})$$

$$J(\theta) = \frac{1}{2n} (X\theta - \vec{y})^T (X\theta - \vec{y})$$

$$X = \begin{bmatrix} - & (x^{(1)})^T & - \\ & \vdots & \\ - & (x^{(n)})^T & - \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

单变量房价预测问题

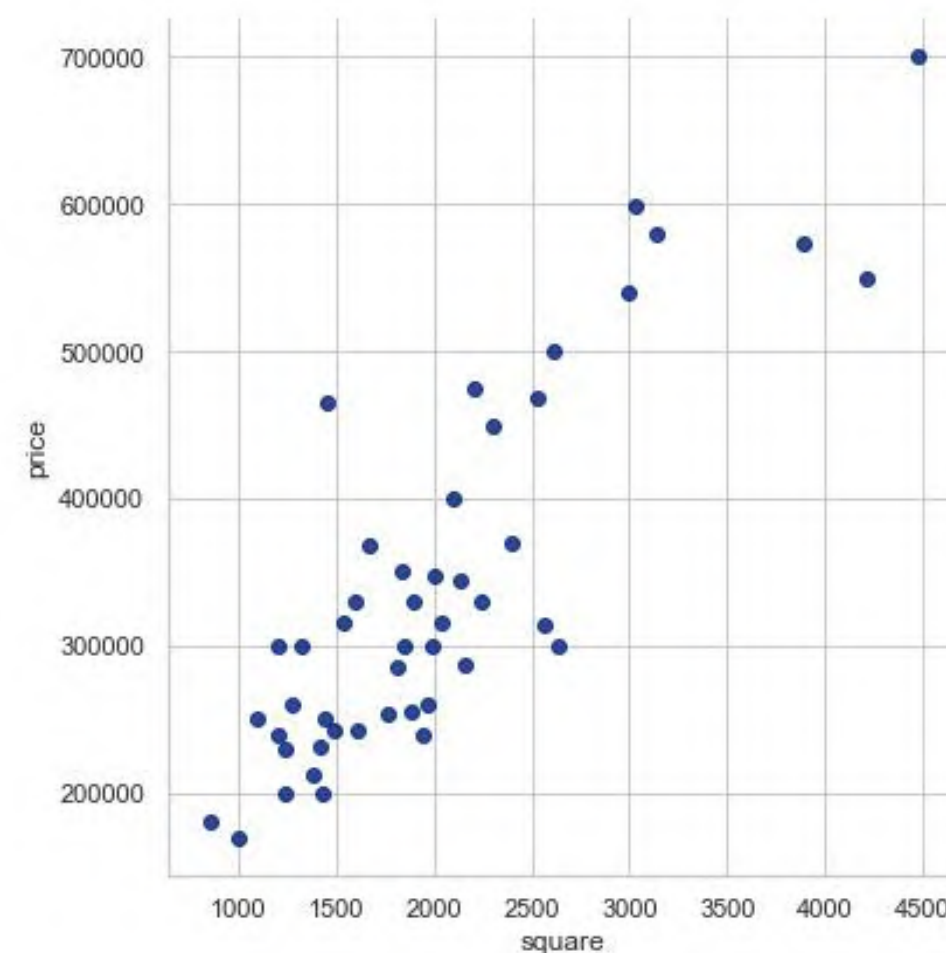
问题描述：根据房屋面积 x 预测其销售价格 y

假设函数：

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 = \theta^T x$$

训练数据：

面积（平方英尺）	价格（美元）
2104	399900
1600	329900
2400	369000
1416	232000
3000	539900
1985	299900
1534	314900
1427	198999
1380	212000
1494	242500



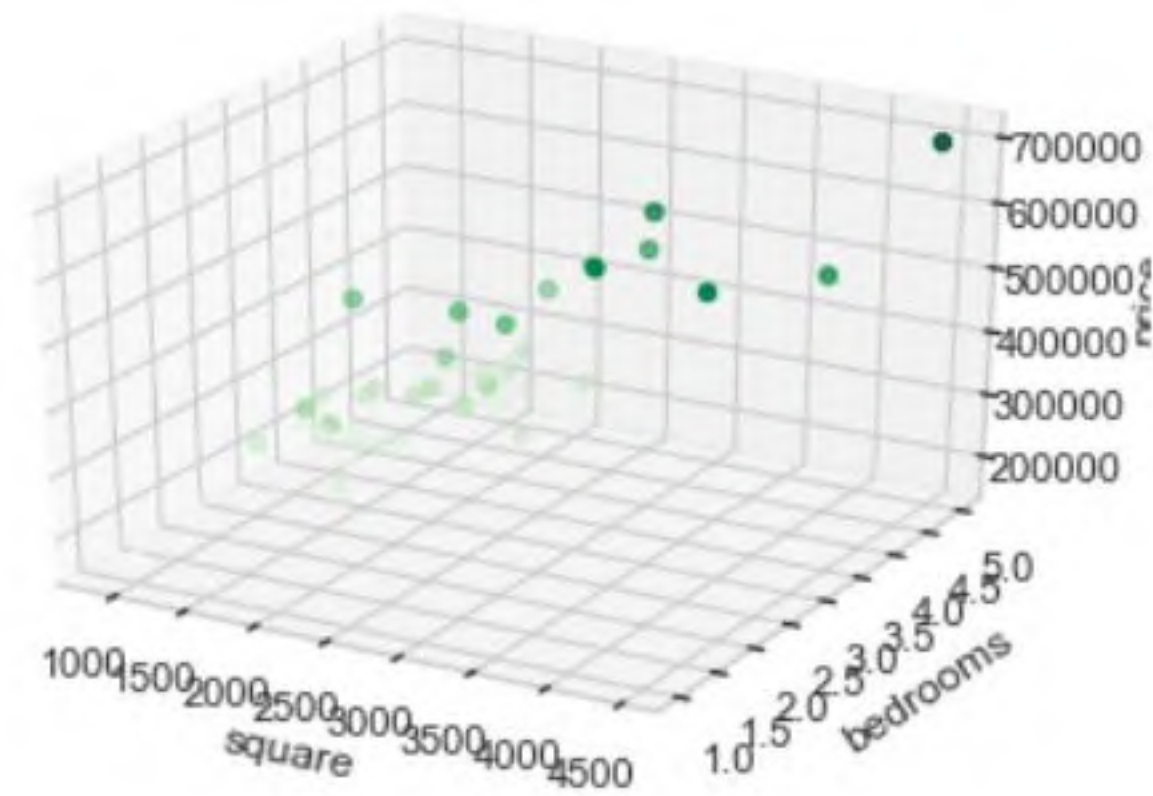
多变量房价预测问题：数据分析

问题描述：根据房屋面积 x_1 和卧室数量 x_2 ，预测其销售价格 y

训练数据：

面积（平方英尺）	卧室数量（个）	价格（美元）
2104	3	399900
1600	3	329900
2400	3	369000
1416	2	232000
3000	4	539900
1985	4	299900
1534	3	314900
1427	3	198999
1380	3	212000
1494	3	242500

数据分布：

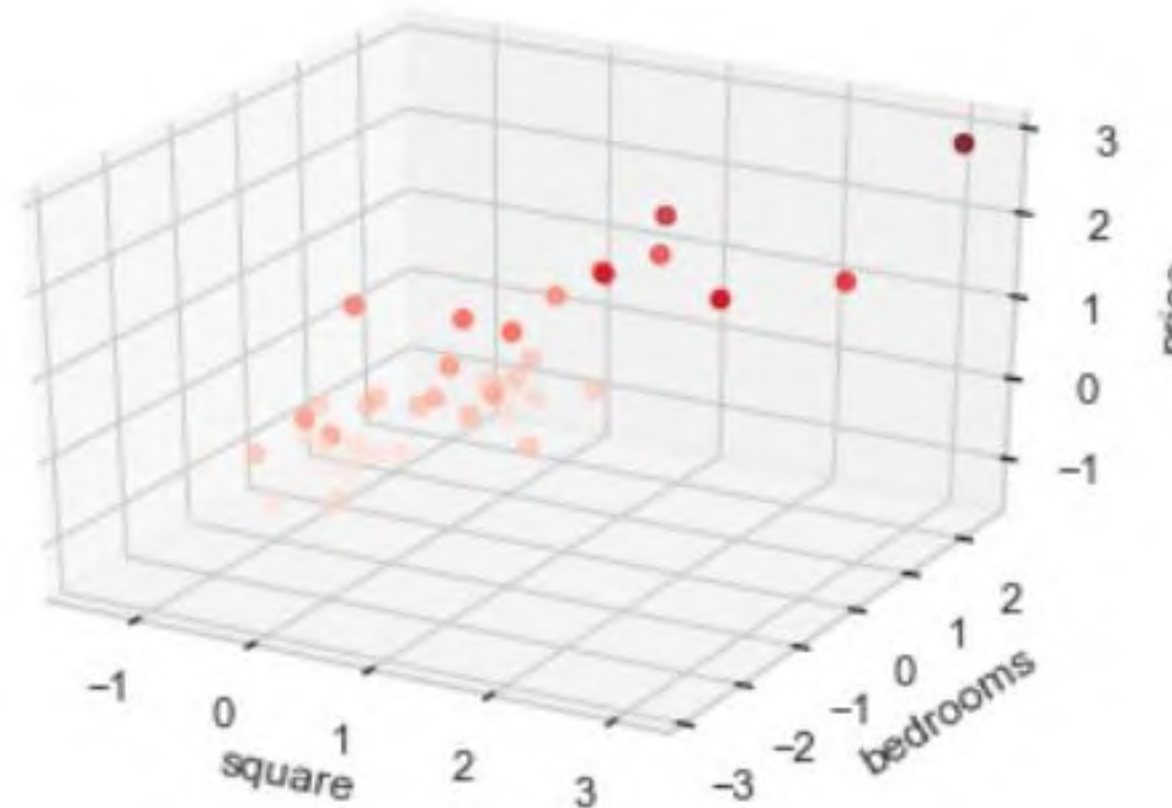
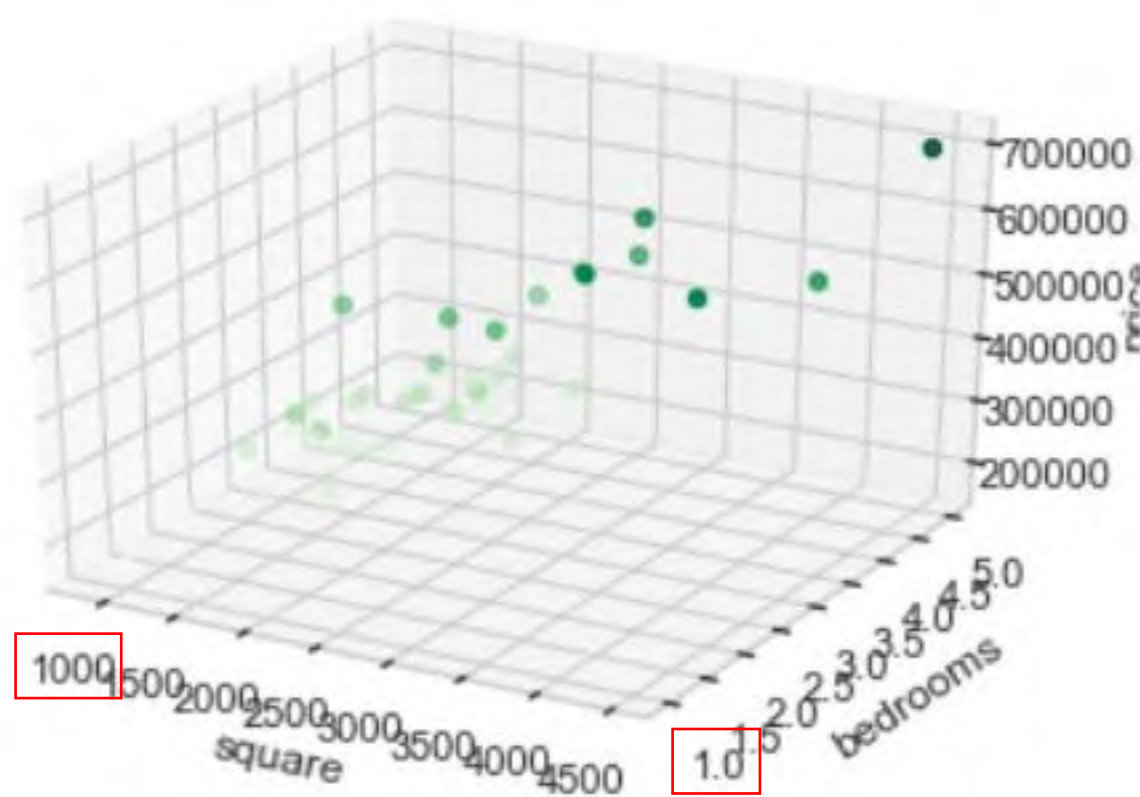


多变量房价预测问题：特征归一化

房屋面积和卧室数量这两个变量（特征）在数值上差了**1000**倍。在这种情况下，通常先进行**特征缩放**（Scaling），再开始训练，可以加速模型收敛。

$$x' = \frac{x - \bar{x}}{\sigma}$$

← 平均值
← 标准差



多变量房价预测问题

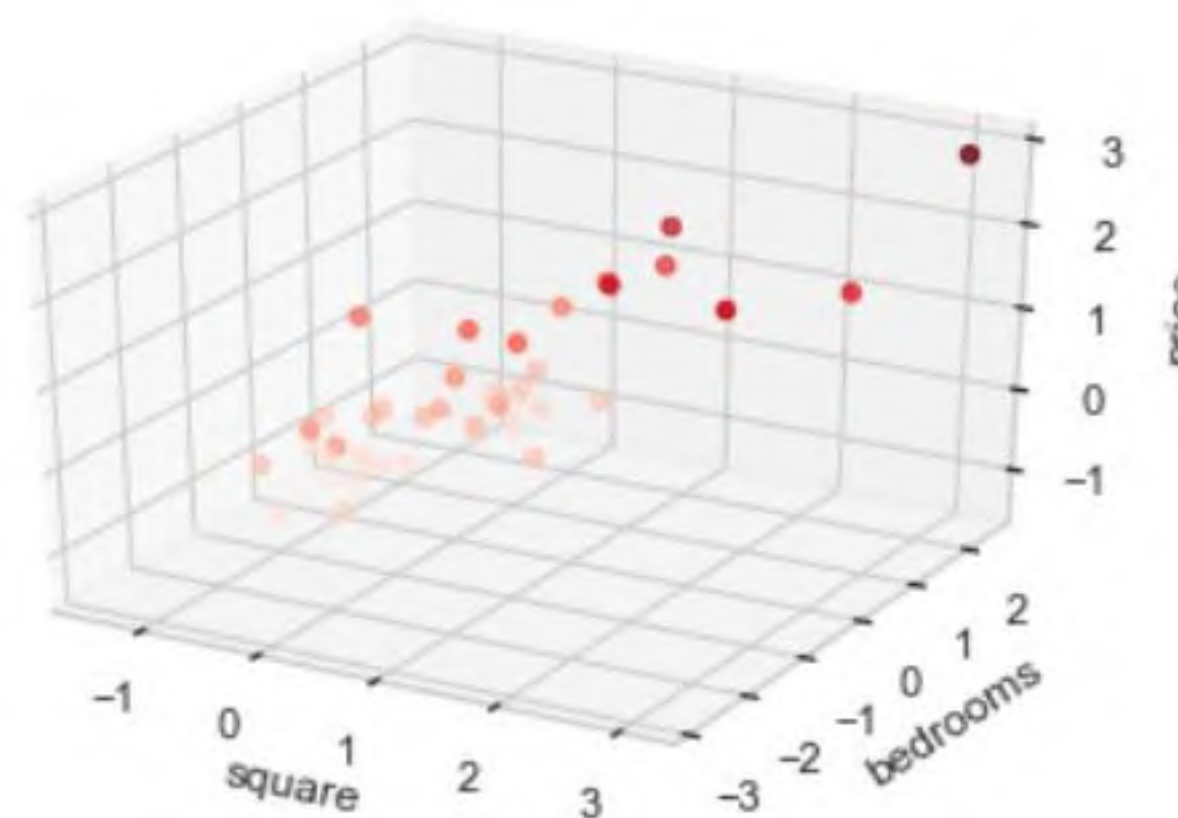
问题描述：根据房屋面积 x_1 和卧室数量 x_2 ，预测其销售价格 y

训练数据：

面积（平方英尺）	卧室数量（个）	价格（美元）
0.13001	-0.22368	0.475747
-0.50419	-0.22368	-0.08407
0.502476	-0.22368	0.228626
-0.73572	-1.53777	-0.86703
1.257476	1.090417	1.595389
-0.01973	1.090417	-0.324
-0.58724	-0.22368	-0.20404
-0.72188	-0.22368	-1.13095
-0.78102	-0.22368	-1.02697
-0.63757	-0.22368	-0.78305

假设函数：

$$h_{\theta}(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = \theta^T X$$



使用 TensorFlow 实现房价预测模型

使用 TensorFlow 训练模型的工作流



数据分析库：Pandas

[Pandas](#) 是一个 [BSD](#) 开源协议许可的，面向 Python 用户的高性能和易于上手的数据结构化和数据分析工具。

数据框（Data Frame）是一个二维带标记的数据结构，每列（column）数据类型可以不同。我们可以将其当作电子表格或数据库表。

	square	bedrooms	price
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

数据读入

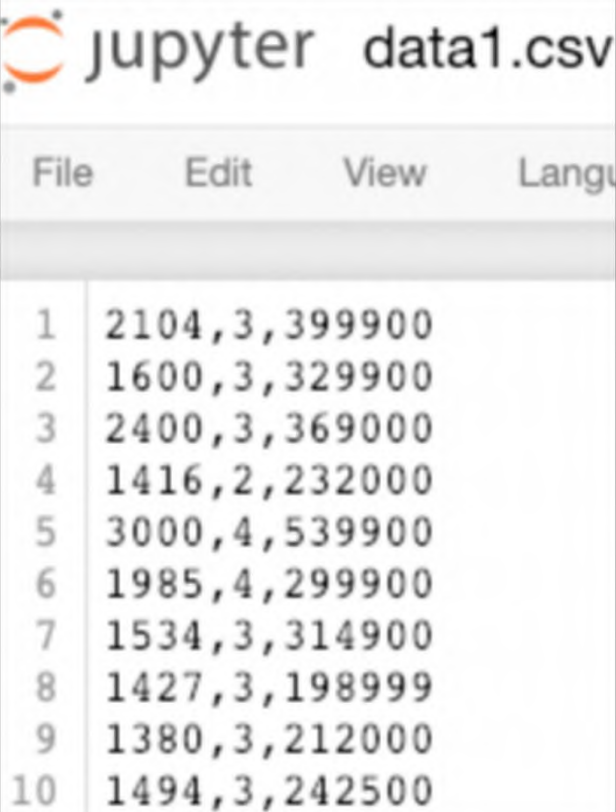
[pandas.read_csv](#) 方法实现了快速读取 CSV（comma-separated）文件到数据框的功能。

```
In [3]: import pandas as pd

df1 = pd.read_csv('data1.csv', names=['square', 'bedrooms', 'price'])
df1.head()
```

Out[3]:

	square	bedrooms	price
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900



jupyter data1.csv	
File	Edit View Language
1	2104,3,399900
2	1600,3,329900
3	2400,3,369000
4	1416,2,232000
5	3000,4,539900
6	1985,4,299900
7	1534,3,314900
8	1427,3,198999
9	1380,3,212000
10	1494,3,242500

数据可视化库：matplotlib & seaborn & mplot3d

[matplotlib](#) 是一个 Python 2D 绘图库，可以生成出版物质量级别的图像和各种硬拷贝格式，并广泛支持多种平台，如：Python 脚本，Python，IPython Shell 和 Jupyter Notebook。

[seaborn](#) 是一个基于 matplotlib 的 Python 数据可视化库。它提供了更易用的高级接口，用于绘制精美且信息丰富的统计图形。

[mpl_toolkits.mplot3d](#) 是一个基础 3D 绘图（散点图、平面图、折线图 etc）工具集，也是 matplotlib 库的一部分。同时，它也支持轻量级的独立安装模式。

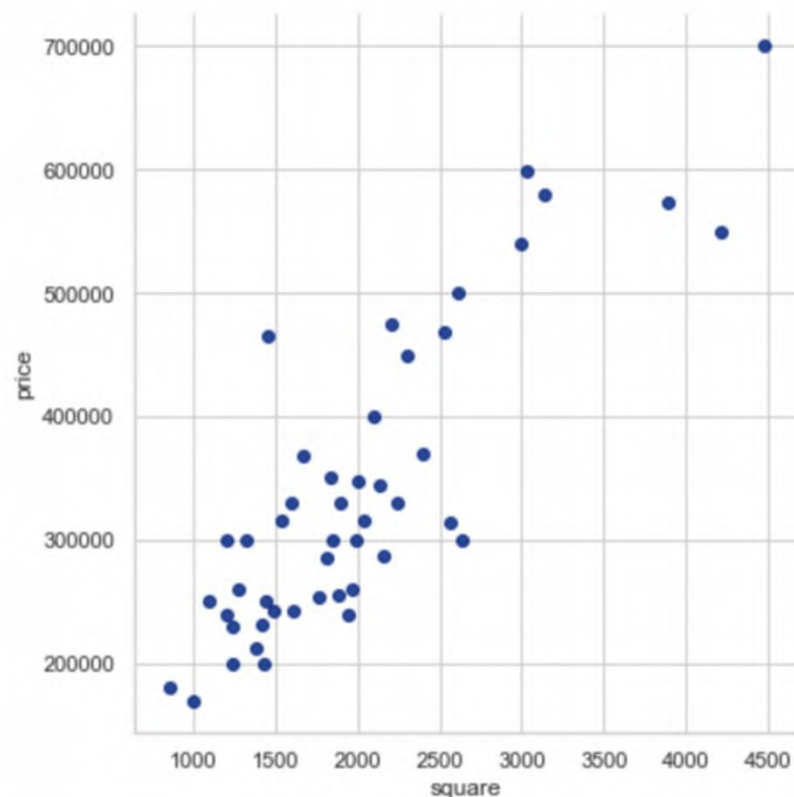
数据分析（2D）

[seaborn.lmplot](#) 方法专门用于线性关系的可视化，适用于回归模型。

```
In [2]: import pandas as pd
import seaborn as sns
sns.set(context="notebook", style="whitegrid", palette="dark")

df0 = pd.read_csv('data0.csv', names=['square', 'price'])
sns.lmplot('square', 'price', df0, height=6, fit_reg=False)
```

Out[2]: <seaborn.axisgrid.FacetGrid at 0x122f090b8>



	square	price
0	2104	399900
1	1600	329900
2	2400	369000
3	1416	232000
4	3000	539900
5	1985	299900
6	1534	314900
7	1427	198999
8	1380	212000
9	1494	242500
10	1940	239999
11	2000	347000
12	1890	329999

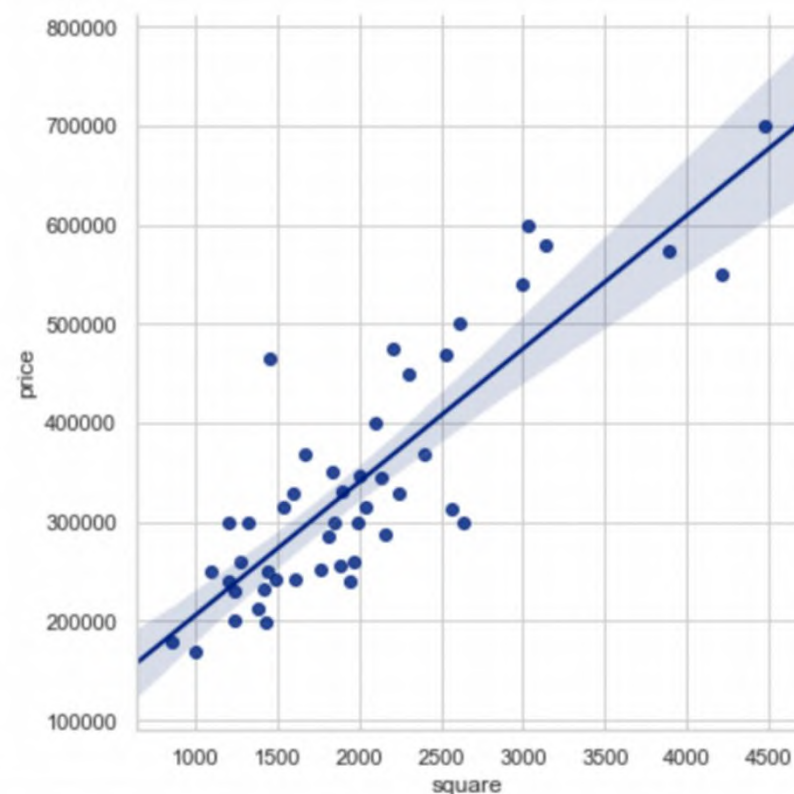
数据分析（2D）

[seaborn.lmplot](#) 方法专门用于线性关系的可视化，适用于回归模型。

```
In [5]: import pandas as pd
import seaborn as sns
sns.set(context="notebook", style="whitegrid", palette="dark")

df0 = pd.read_csv('data0.csv', names=['square', 'price'])
sns.lmplot('square', 'price', df0, height=6, fit_reg=True)
```

Out[5]: <seaborn.axisgrid.FacetGrid at 0x1235da588>



	square	price
0	2104	399900
1	1600	329900
2	2400	369000
3	1416	232000
4	3000	539900
5	1985	299900
6	1534	314900
7	1427	198999
8	1380	212000
9	1494	242500
10	1940	239999
11	2000	347000
12	1890	329999

数据分析（3D）

[Axes3D.scatter3D](#) 方法专门用于绘制3维的散点图。

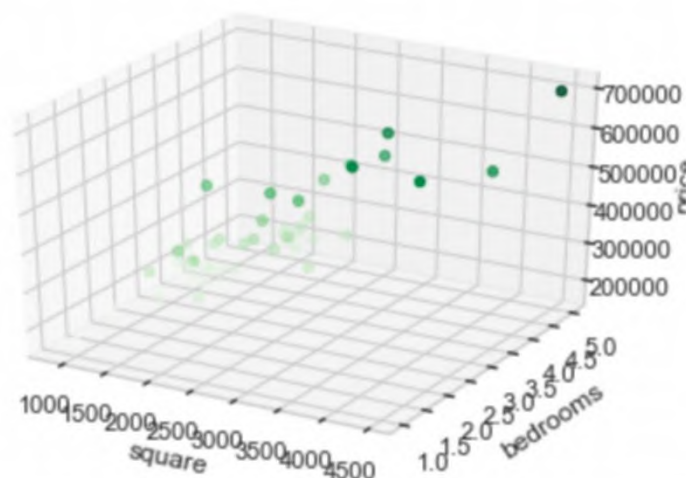
```
In [4]: from mpl_toolkits import mplot3d

import pandas as pd
import matplotlib.pyplot as plt

df1 = pd.read_csv('data1.csv', names=['square', 'bedrooms', 'price'])
df1.head()

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.set_xlabel('square')
ax.set_ylabel('bedrooms')
ax.set_zlabel('price')
ax.scatter3D(df1['square'], df1['bedrooms'], df1['price'], c=df1['price'], cmap='Greens')
```

Out[4]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x123207ac8>



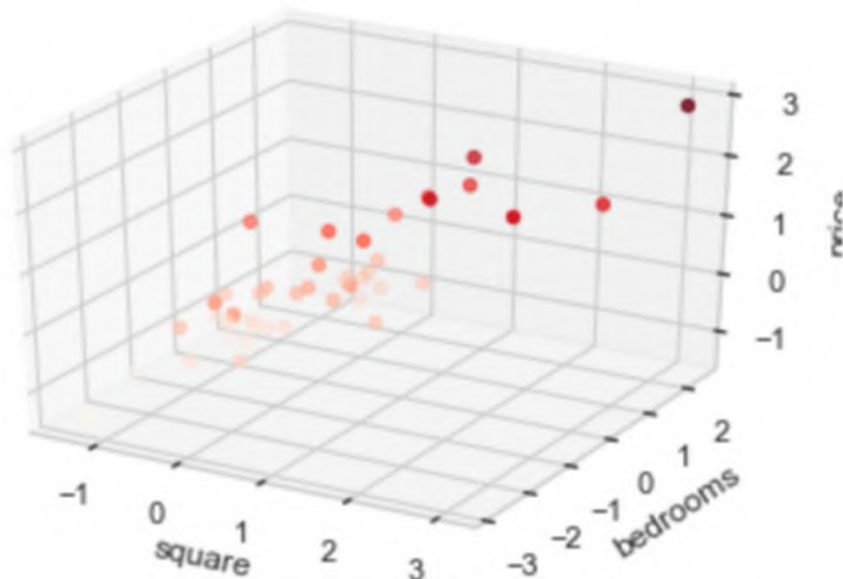
	square	bedrooms	price
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900
5	1985	4	299900
6	1534	3	314900
7	1427	3	198999
8	1380	3	212000
9	1494	3	242500
10	1940	4	239999
11	2000	3	347000
12	1890	3	329999
13	4478	5	699900
14	1268	3	259900
15	2300	4	449900

数据归一化（3D）

```
In [7]: def normalize_feature(df):
        return df.apply(lambda column: (column - column.mean()) / column.std())

df = normalize_feature(df1)
ax = plt.axes(projection='3d')
ax.set_xlabel('square')
ax.set_ylabel('bedrooms')
ax.set_zlabel('price')
ax.scatter3D(df['square'], df['bedrooms'], df['price'], c=df['price'], cmap='Reds')
```

Out[7]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x12323d3c8>



	square	bedrooms	price
0	0.130010	-0.223675	0.475747
1	-0.504190	-0.223675	-0.084074
2	0.502476	-0.223675	0.228626
3	-0.735723	-1.537767	-0.867025
4	1.257476	1.090417	1.595389
5	-0.019732	1.090417	-0.323998
6	-0.587240	-0.223675	-0.204036
7	-0.721881	-0.223675	-1.130948
8	-0.781023	-0.223675	-1.026973
9	-0.637573	-0.223675	-0.783051
10	-0.076357	1.090417	-0.803053
11	-0.000857	-0.223675	0.052682
12	-0.139273	-0.223675	-0.083283
13	3.117292	2.404508	2.874981
14	-0.921956	-0.223675	-0.643896
15	0.376643	1.090417	0.875619

数据处理: NumPy

[NumPy](#) 是一个 [BSD](#) 开源协议许可的, 面向 Python 用户的基础科学计算库, 在多维数组上实现了线性代数、傅立叶变换和其他丰富的函数运算。

X y

```
In [2]: import pandas as pd
import numpy as np

def normalize_feature(df):
    return df.apply(lambda column: (column - column.mean()) / column.std())

df = normalize_feature(pd.read_csv('data1.csv',
                                   names=['square', 'bedrooms', 'price']))

ones = pd.DataFrame({'ones': np.ones(len(df))}) # ones是n行1列的数据框, 表示x0恒为1
df = pd.concat([ones, df], axis=1) # 根据列合并数据

X_data = np.array(df[df.columns[0:3]])
y_data = np.array(df[df.columns[-1]]).reshape(len(df), 1)

print(X_data.shape, type(X_data))
print(y_data.shape, type(y_data))

(47, 3) <class 'numpy.ndarray'>
(47, 1) <class 'numpy.ndarray'>
```

	ones	square	bedrooms	price
0	1.0	0.130010	-0.223675	0.475747
1	1.0	-0.504190	-0.223675	-0.084074
2	1.0	0.502476	-0.223675	0.228626
3	1.0	-0.735723	-1.537767	-0.867025
4	1.0	1.257476	1.090417	1.595389
5	1.0	-0.019732	1.090417	-0.323998
6	1.0	-0.587240	-0.223675	-0.204036
7	1.0	-0.721881	-0.223675	-1.130948
8	1.0	-0.781023	-0.223675	-1.026973
9	1.0	-0.637573	-0.223675	-0.783051
10	1.0	-0.076357	1.090417	-0.803053
11	1.0	-0.000857	-0.223675	0.052682
12	1.0	-0.139273	-0.223675	-0.083283
13	1.0	3.117292	2.404508	2.874981
14	1.0	-0.921956	-0.223675	-0.643896

创建线性回归模型（数据流图）

```
In [2]: import tensorflow as tf

alpha = 0.01 # 学习率 alpha
epoch = 500 # 训练全量数据集的轮数

# 创建线性回归模型（数据流图）
# 输入 X, 形状[47, 3]
X = tf.placeholder(tf.float32, X_data.shape)
# 输出 y, 形状[47, 1]
y = tf.placeholder(tf.float32, y_data.shape)

# 权重变量 W, 形状[3,1]
W = tf.get_variable("weights", (X_data.shape[1], 1), initializer=tf.constant_initializer())

# 假设函数  $h(x) = w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2$ , 其中  $x_0$  恒为1
# 推理值 y_pred 形状[47,1]
y_pred = tf.matmul(X, W)

# 损失函数采用最小二乘法, y_pred - y 是形如[47, 1]的向量。
# tf.matmul(a,b,transpose_a=True) 表示: 矩阵a的转置乘矩阵b, 即 [1,47] X [47,1]
# 损失函数操作 loss
loss_op = 1 / (2 * len(X_data)) * tf.matmul((y_pred - y), (y_pred - y), transpose_a=True)
# 随机梯度下降优化器 opt
opt = tf.train.GradientDescentOptimizer(learning_rate=alpha)
# 单步训练操作 train_op
train_op = opt.minimize(loss_op)
```

创建会话（运行环境）

```
In [4]: # 创建会话 (运行环境)
with tf.Session() as sess:
    # 初始化全局变量
    sess.run(tf.global_variables_initializer())
    # 开始训练模型
    # 因为训练集较小, 所以采用批梯度下降优化算法, 每次都使用全量数据训练
    for e in range(1, epoch + 1):
        sess.run(train_op, feed_dict={X: X_data, y: y_data})
        if e % 10 == 0:
            loss, w = sess.run([loss_op, W], feed_dict={X: X_data, y: y_data})
            log_str = "Epoch %d \t Loss=%.4g \t Model: y = %.4gx1 + %.4gx2 + %.4g"
            print(log_str % (e, loss, w[1], w[2], w[0]))
```

Epoch 10	Loss=0.4116	Model: y = 0.0791x1 + 0.03948x2 + 3.353e-10
Epoch 20	Loss=0.353	Model: y = 0.1489x1 + 0.07135x2 + -5.588e-11
Epoch 30	Loss=0.3087	Model: y = 0.2107x1 + 0.09676x2 + 3.912e-10
Epoch 40	Loss=0.2748	Model: y = 0.2655x1 + 0.1167x2 + -1.863e-11
Epoch 50	Loss=0.2489	Model: y = 0.3142x1 + 0.1321x2 + 1.77e-10
Epoch 60	Loss=0.2288	Model: y = 0.3576x1 + 0.1436x2 + -4.47e-10
Epoch 70	Loss=0.2131	Model: y = 0.3965x1 + 0.1519x2 + -8.103e-10
Epoch 80	Loss=0.2007	Model: y = 0.4313x1 + 0.1574x2 + -6.985e-10
Epoch 90	Loss=0.1908	Model: y = 0.4626x1 + 0.1607x2 + -4.936e-10
Epoch 100	Loss=0.1828	Model: y = 0.4909x1 + 0.1621x2 + -6.147e-10
Epoch 110	Loss=0.1763	Model: y = 0.5165x1 + 0.162x2 + -7.87e-10
Epoch 120	Loss=0.1709	Model: y = 0.5397x1 + 0.1606x2 + -5.821e-10
Epoch 130	Loss=0.1664	Model: y = 0.5609x1 + 0.1581x2 + -9.08e-10
Epoch 140	Loss=0.1625	Model: y = 0.5802x1 + 0.1549x2 + -9.965e-10
Epoch 150	Loss=0.1592	Model: y = 0.5979x1 + 0.1509x2 + -9.756e-10
Epoch 160	Loss=0.1564	Model: y = 0.6142x1 + 0.1465x2 + -4.144e-10
Epoch 170	Loss=0.1539	Model: y = 0.6292x1 + 0.1416x2 + -1.001e-10
Epoch 180	Loss=0.1518	Model: y = 0.643x1 + 0.1364x2 + -3.236e-10
Epoch 190	Loss=0.1498	Model: y = 0.6559x1 + 0.131x2 + -6.286e-11
Epoch 200	Loss=0.1481	Model: y = 0.6678x1 + 0.1255x2 + 2.119e-10

估计模型： $y = 0.8304x_1 + 8.239^{-4}x_2 + 4.138^{-9}$

使用 TensorBoard 可视化模型数据流图

TensorBoard 可视化工具

在数据处理过程中，用户通常想要可视化地直观查看**数据集**分布情况。
在模型设计过程中，用户往往需要分析和检查**数据流图**是否正确实现。
在模型训练过程中，用户也常常需要关注**模型参数**和**超参数**变化趋势。
在模型测试过程中，用户也往往需要查看**准确率**和**召回率**等评估指标。

因此，TensorFlow 项目组开发了机器学习可视化工具 **TensorBoard**，
它通过展示直观的图形，能够有效地辅助机器学习程序的开发者和使用者理解算法模型及其工作流程，提升模型开发工作效率。

SCALARS

CUSTOM SCALARS

IMAGES

AUDIO

DEBUGGER

GRAPHS

DISTRIBUTIONS

HISTOGRAMS

PROJECTOR

TEXT

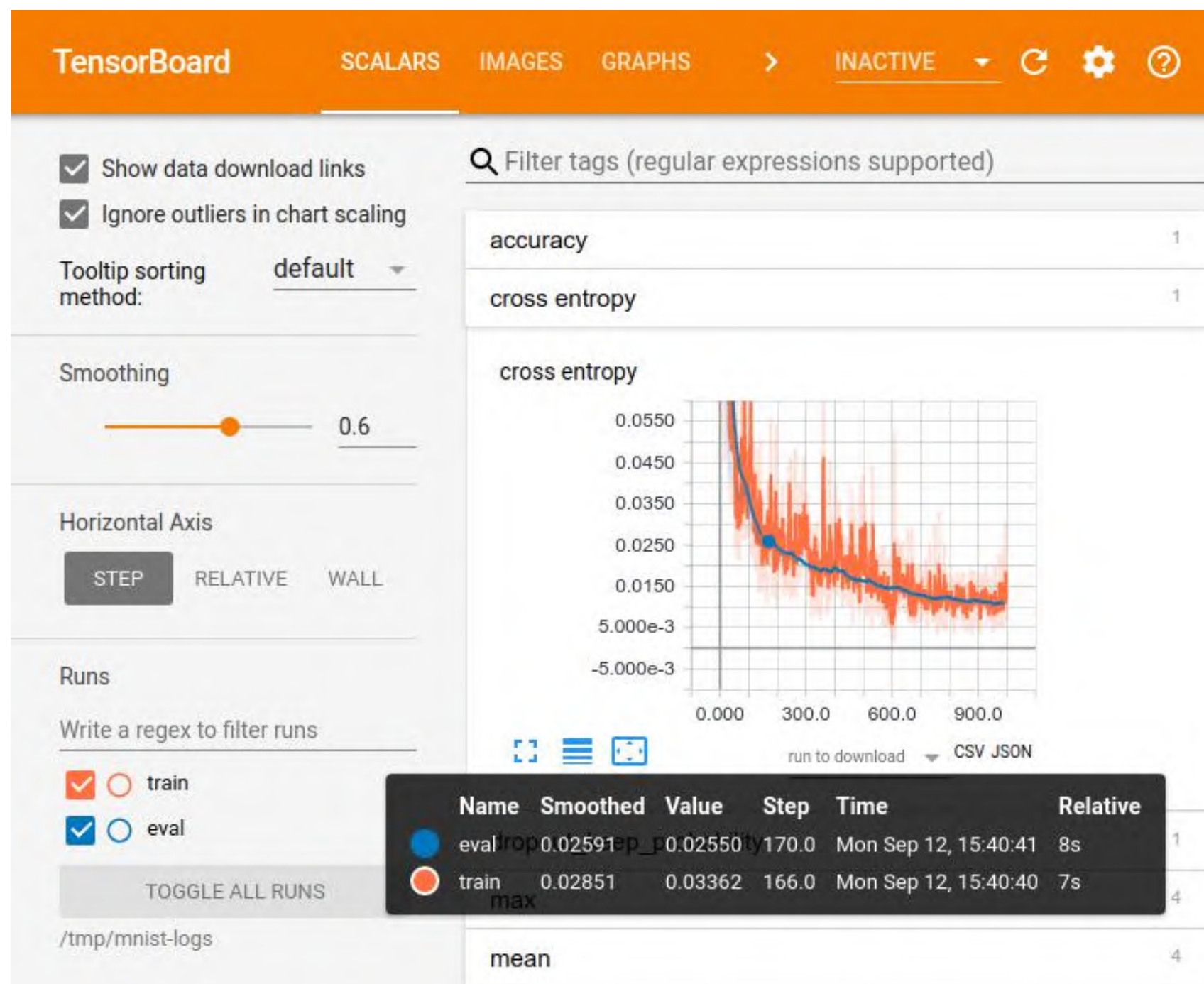
PR CURVES

PROFILE

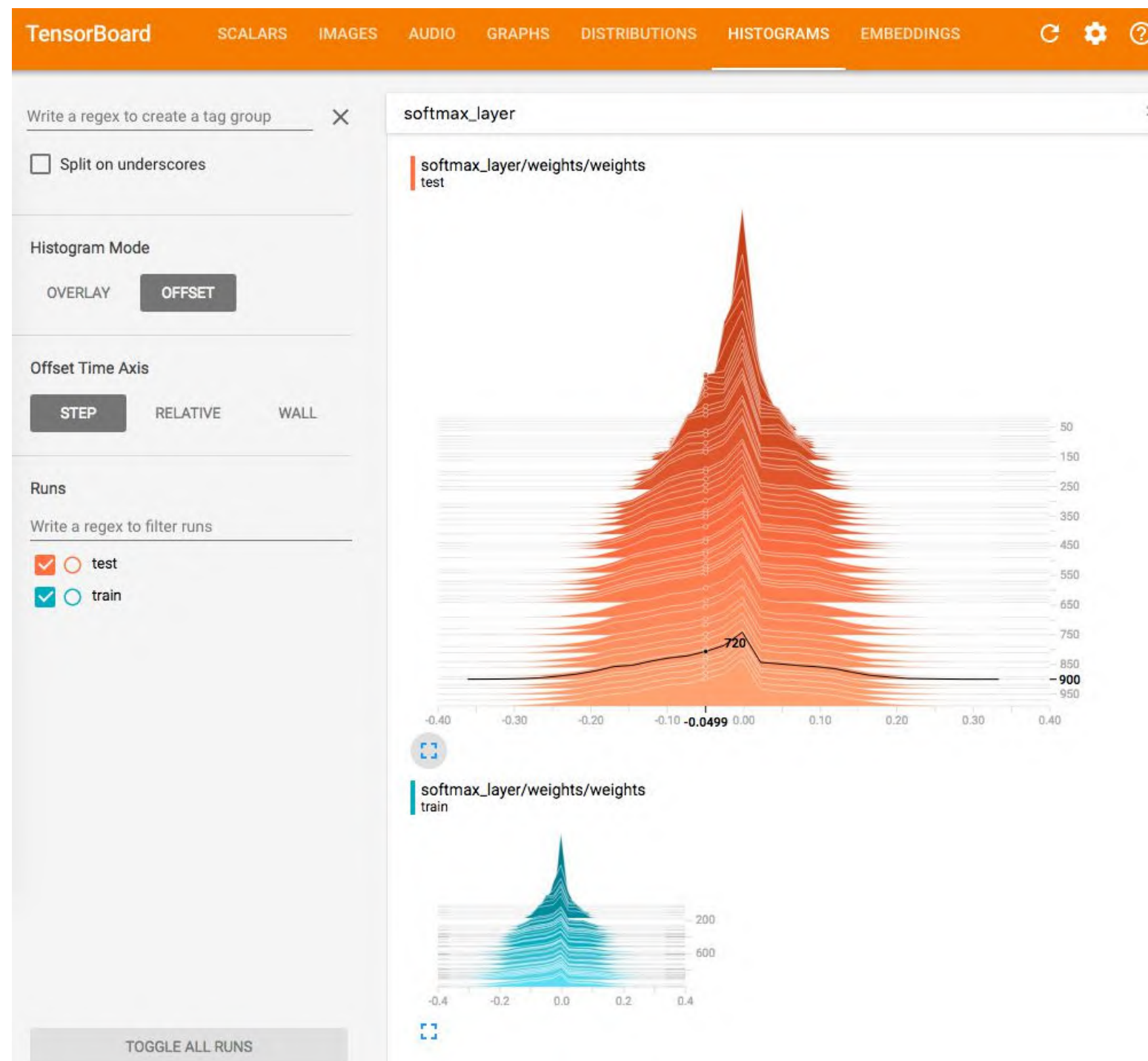
BEHOLDER

WHAT-IF TOOL

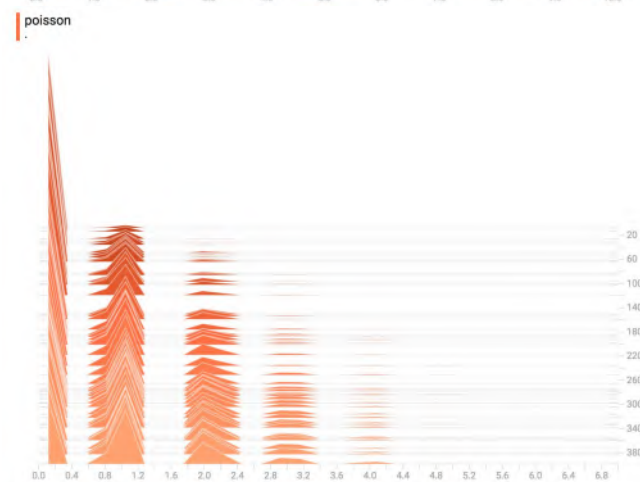
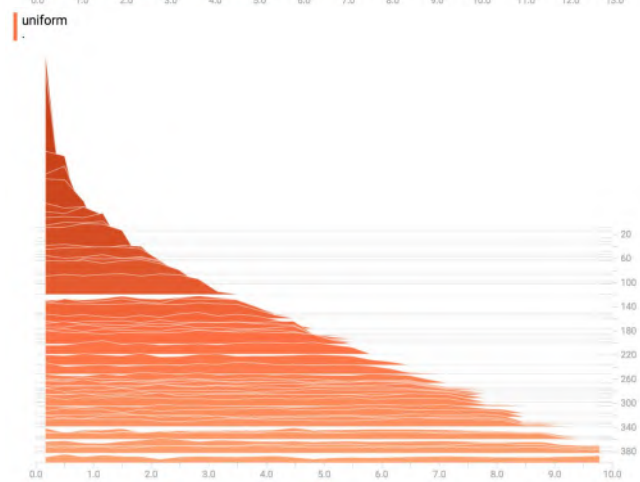
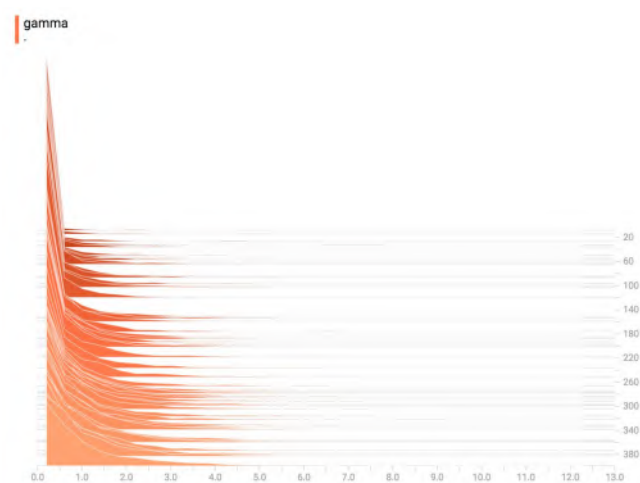
TensorBoard 可视化训练



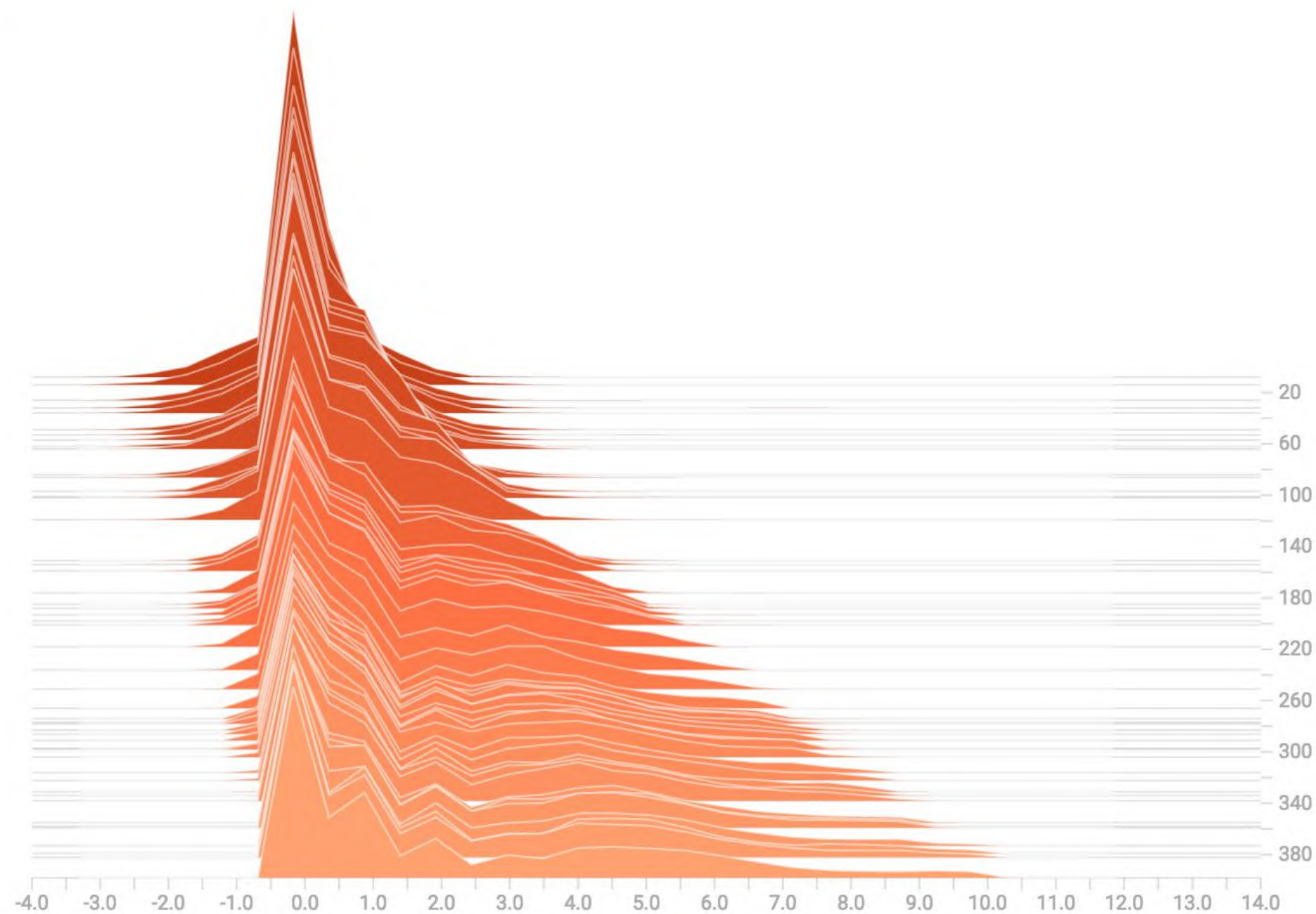
TensorBoard 可视化统计数据



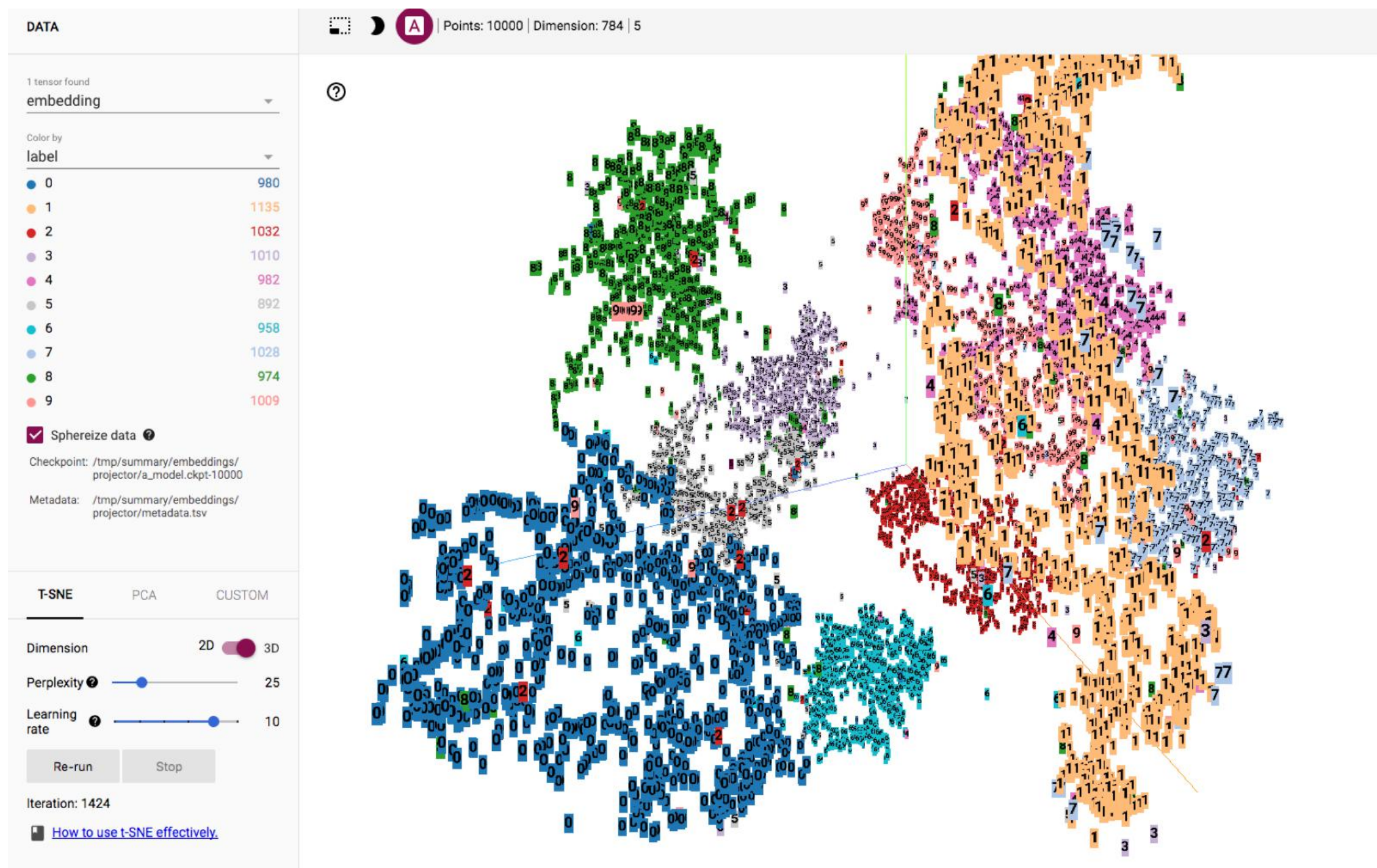
TensorBoard 可视化数据分布



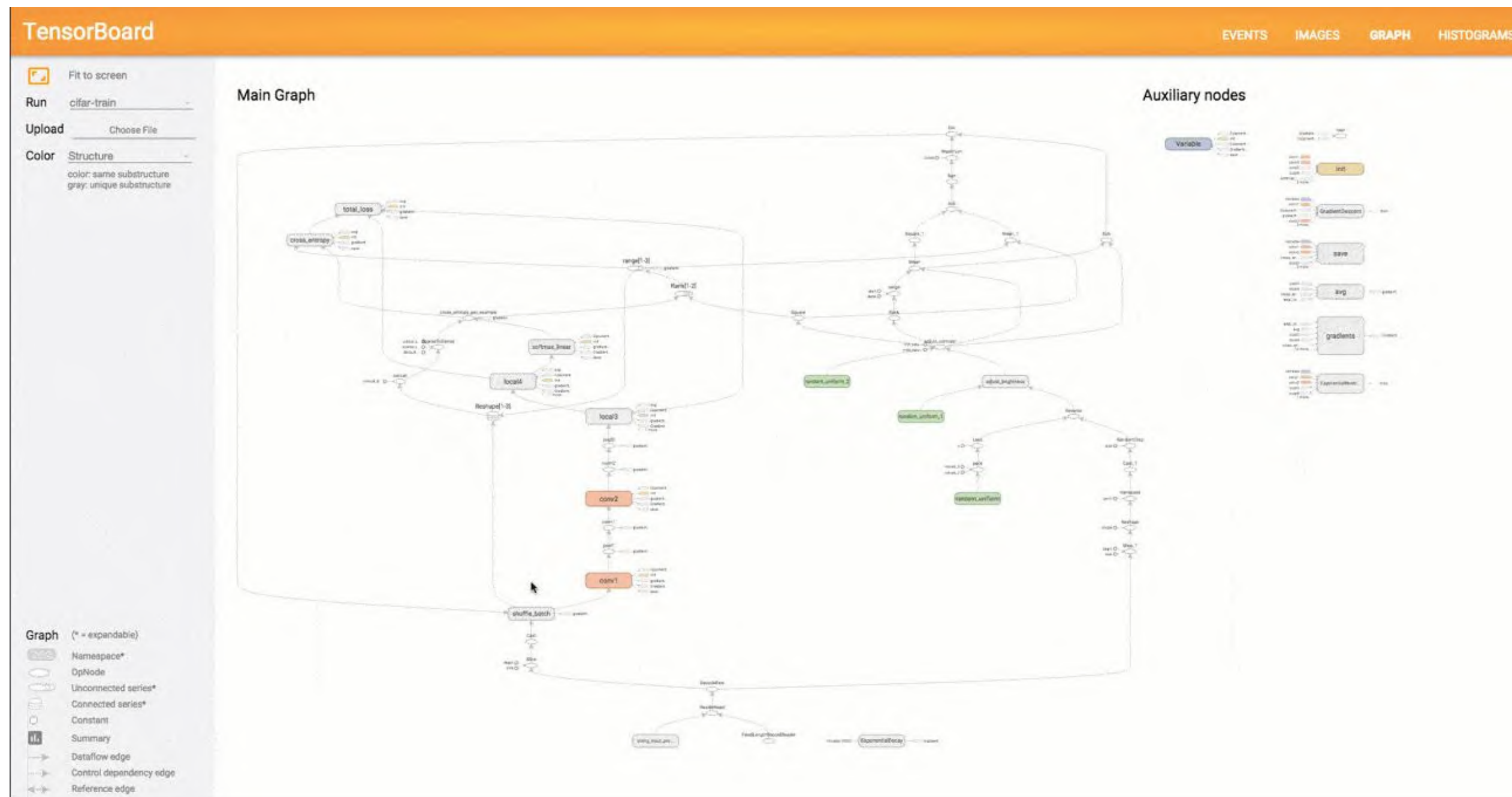
all_combined



TensorBoard 可视化数据集 (MNIST)

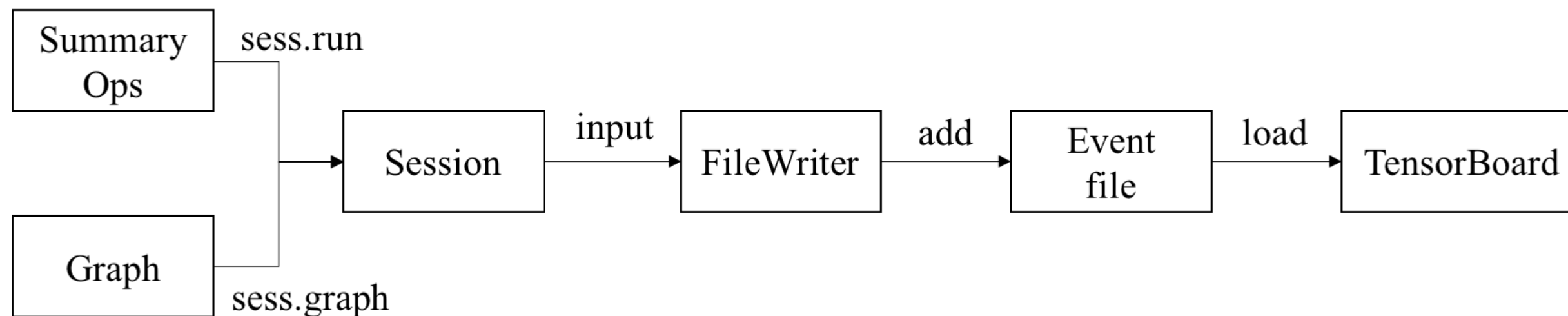


TensorBoard 可视化数据流图



TensorBoard 使用流程

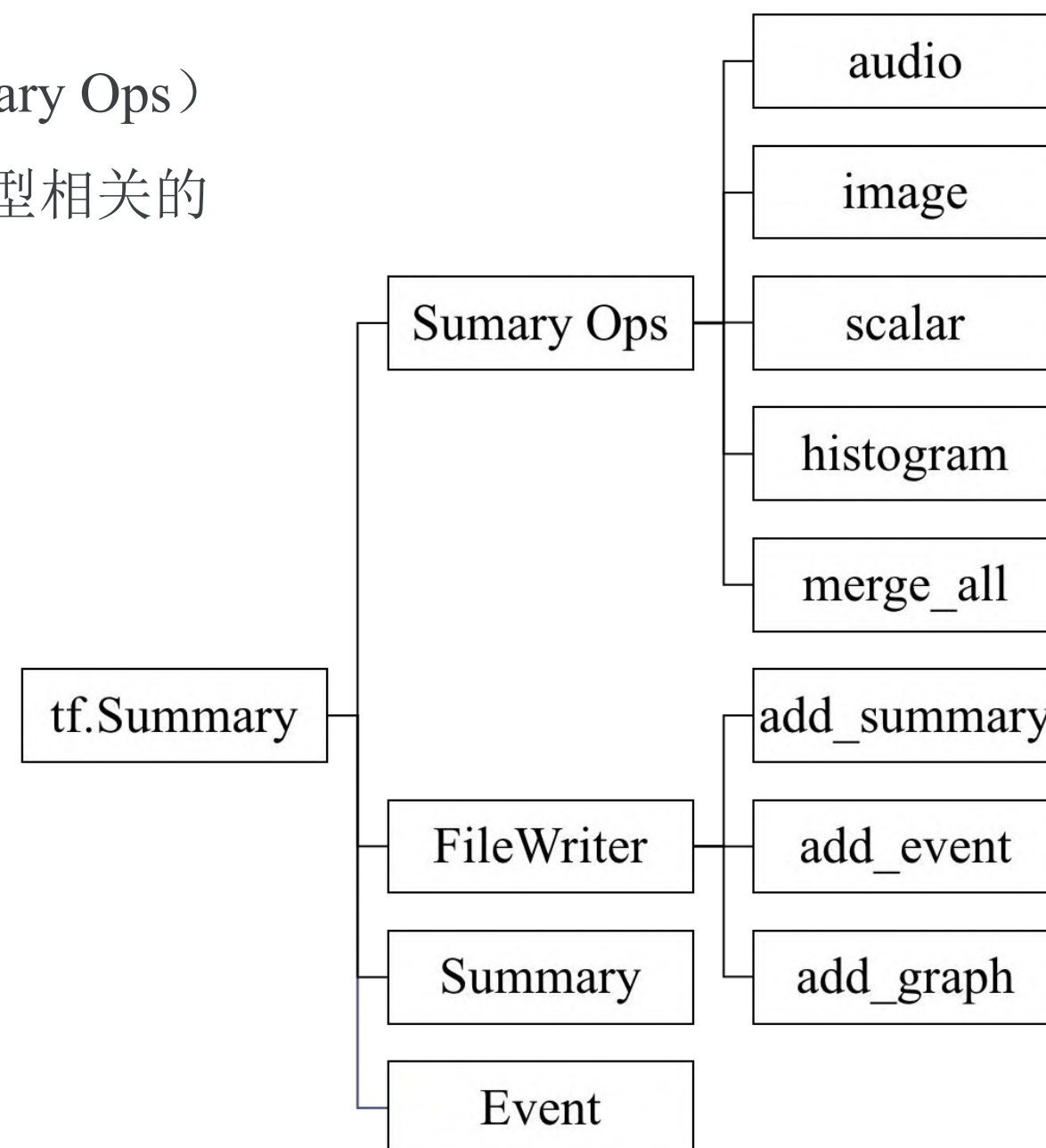
可视化的数据是数据流图和张量，它们需要在会话中加载或执行操作后才能获取。然后，用户需要使用 **FileWriter** 实例将这些数据写入事件文件。最后，启动 TensorBoard 程序，加载事件文件中的序列化数据，从而可以在各个面板中展示对应的可视化对象。



tf.summary 模块介绍

前述流程中使用的 FileWriter 实例和汇总操作（Summary Ops）均属于 tf.summary 模块。其主要功能是获取和输出模型相关的序列化数据，它贯通 TensorBoard 的整个使用流程。

tf.summary 模块的核心部分由一组汇总操作以及 FileWriter、Summary 和 Event 3 个类组成。



可视化数据流图 workflow

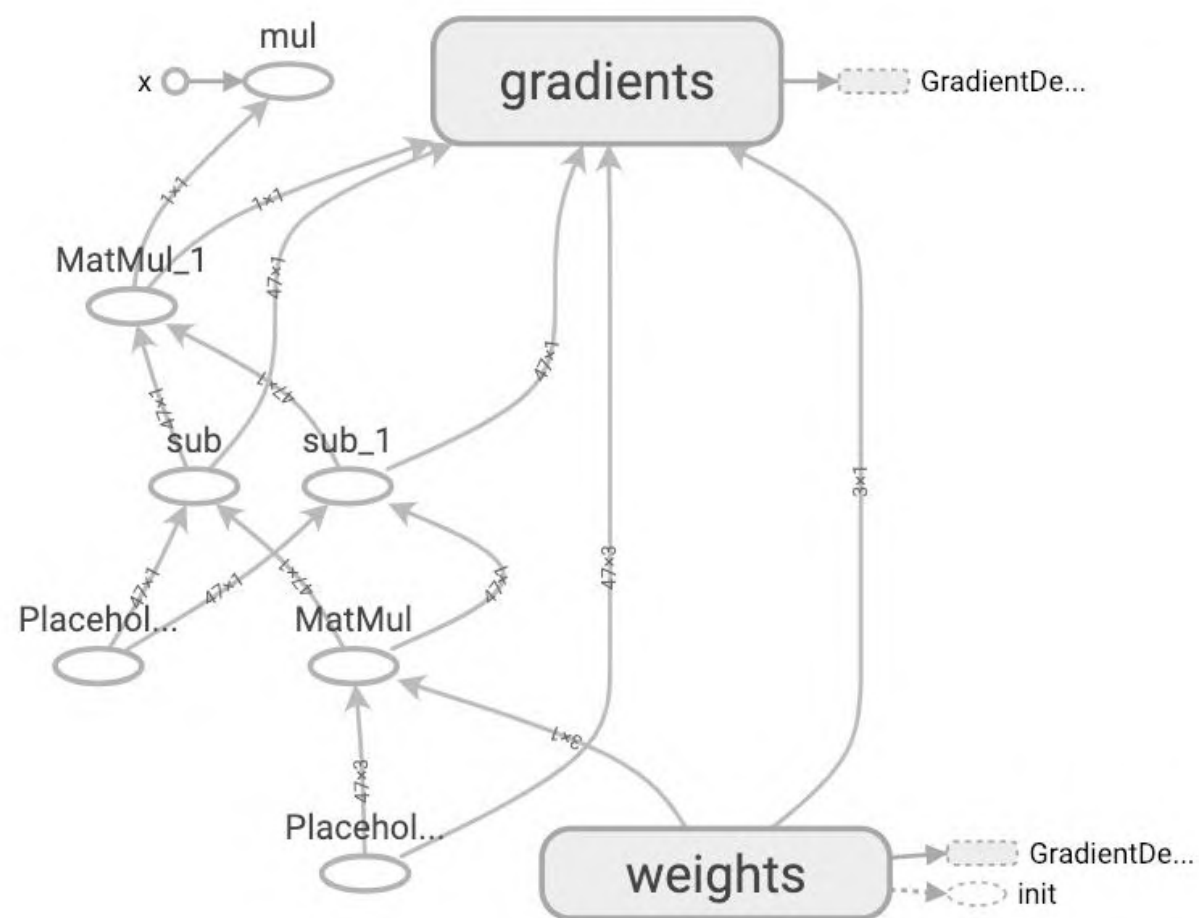
创建
数据流图

创建
FileWriter 实例

启动
TensorBoard

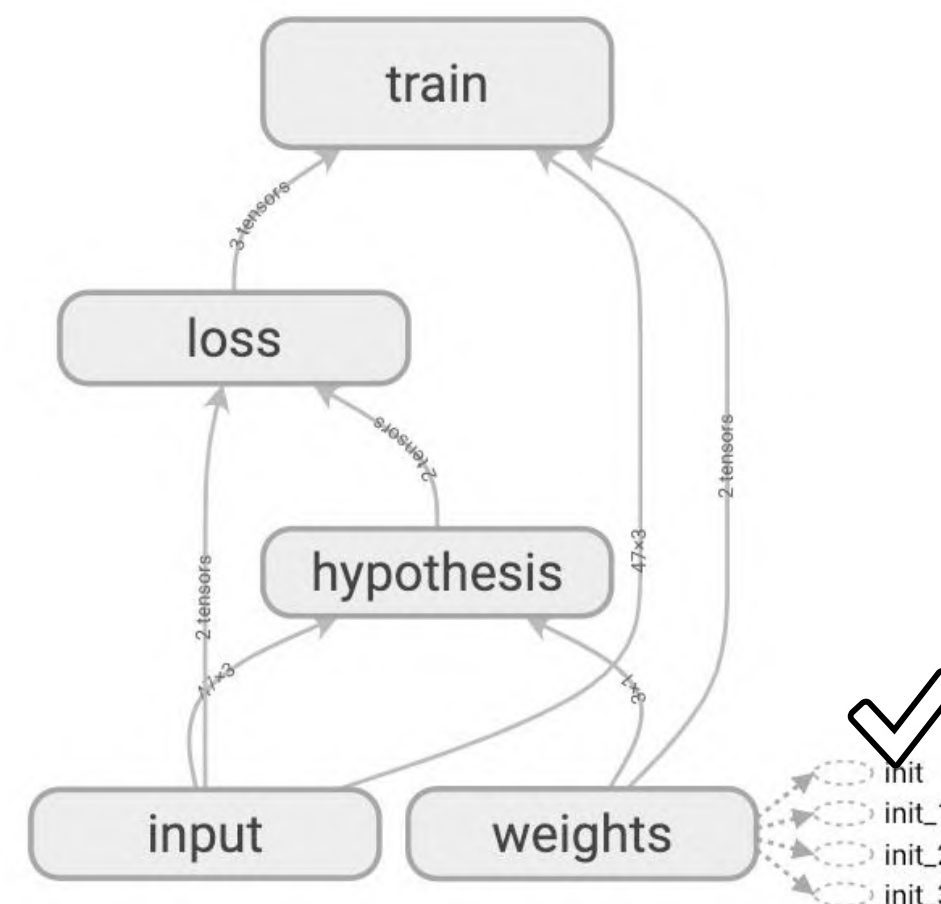
Which one is better?

Main Graph



VS

Main Graph



名字作用域与抽象节点

```
In [2]: import tensorflow as tf

alpha = 0.01 # 学习率 alpha
epoch = 500 # 训练全量数据集的轮数

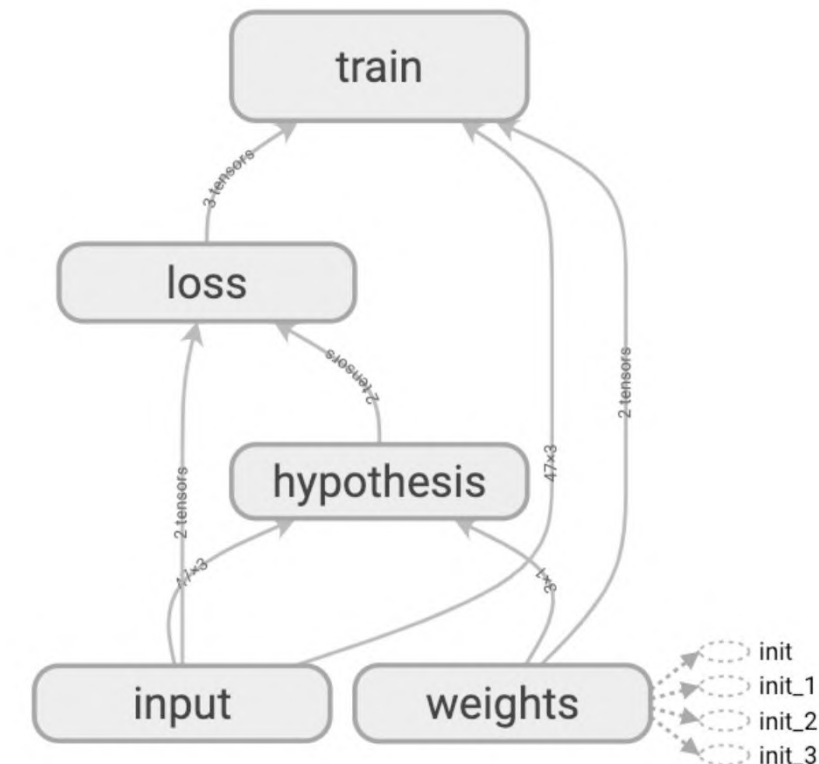
# 创建线性回归模型 (数据流图)
with tf.name_scope('input'):
    # 输入 X, 形状[47, 3]
    X = tf.placeholder(tf.float32, X_data.shape, name='X')
    # 输出 y, 形状[47, 1]
    y = tf.placeholder(tf.float32, y_data.shape, name='y')

with tf.name_scope('hypothesis'):
    # 权重变量 W, 形状[3,1]
    W = tf.get_variable("weights",
                        (X_data.shape[1], 1),
                        initializer=tf.constant_initializer())
    # 假设函数  $h(x) = w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2$ , 其中  $x_0$  恒为1
    # 推理值 y_pred 形状[47,1]
    y_pred = tf.matmul(X, W, name='y_pred')

with tf.name_scope('loss'):
    # 损失函数采用最小二乘法, y_pred - y 是形如[47, 1]的向量。
    # tf.matmul(a,b,transpose_a=True) 表示: 矩阵a的转置乘矩阵b, 即 [1,47] X [47,1]
    # 损失函数操作 loss
    loss_op = 1 / (2 * len(X_data)) * tf.matmul((y_pred - y), (y_pred - y), transpose_a=True)

with tf.name_scope('train'):
    # 随机梯度下降优化器 opt
    train_op = tf.train.GradientDescentOptimizer(learning_rate=alpha).minimize(loss_op)
```

Main Graph



创建 FileWriter 实例

```
In [3]: # 创建会话 (运行环境)
with tf.Session() as sess:
    # 初始化全局变量
    sess.run(tf.global_variables_initializer())
    # 创建FileWriter实例, 并传入当前会话加载的数据流图
    writer = tf.summary.FileWriter('./summary/linear-regression-1', sess.graph)
    # 开始训练模型
    # 因为训练集较小, 所以每轮都使用全量数据训练
    for e in range(1, epoch + 1):
        sess.run(train_op, feed_dict={X: X_data, y: y_data})
        if e % 10 == 0:
            loss, w = sess.run([loss_op, W], feed_dict={X: X_data, y: y_data})
            log_str = "Epoch %d \t Loss=%.4g \t Model: y = %.4gx1 + %.4gx2 + %.4g"
            print(log_str % (e, loss, w[1], w[2], w[0]))

    # 关闭FileWriter的输出流
    writer.close()
```

Files

Running

Clusters

Select items to perform actions on them.

Upload

New ▾

↺

☐ 0 ▾

/ notebook-examples / chapter-4 / summary / linear-regression-1

Name ▾

Last Modified

File size

..

seconds ago

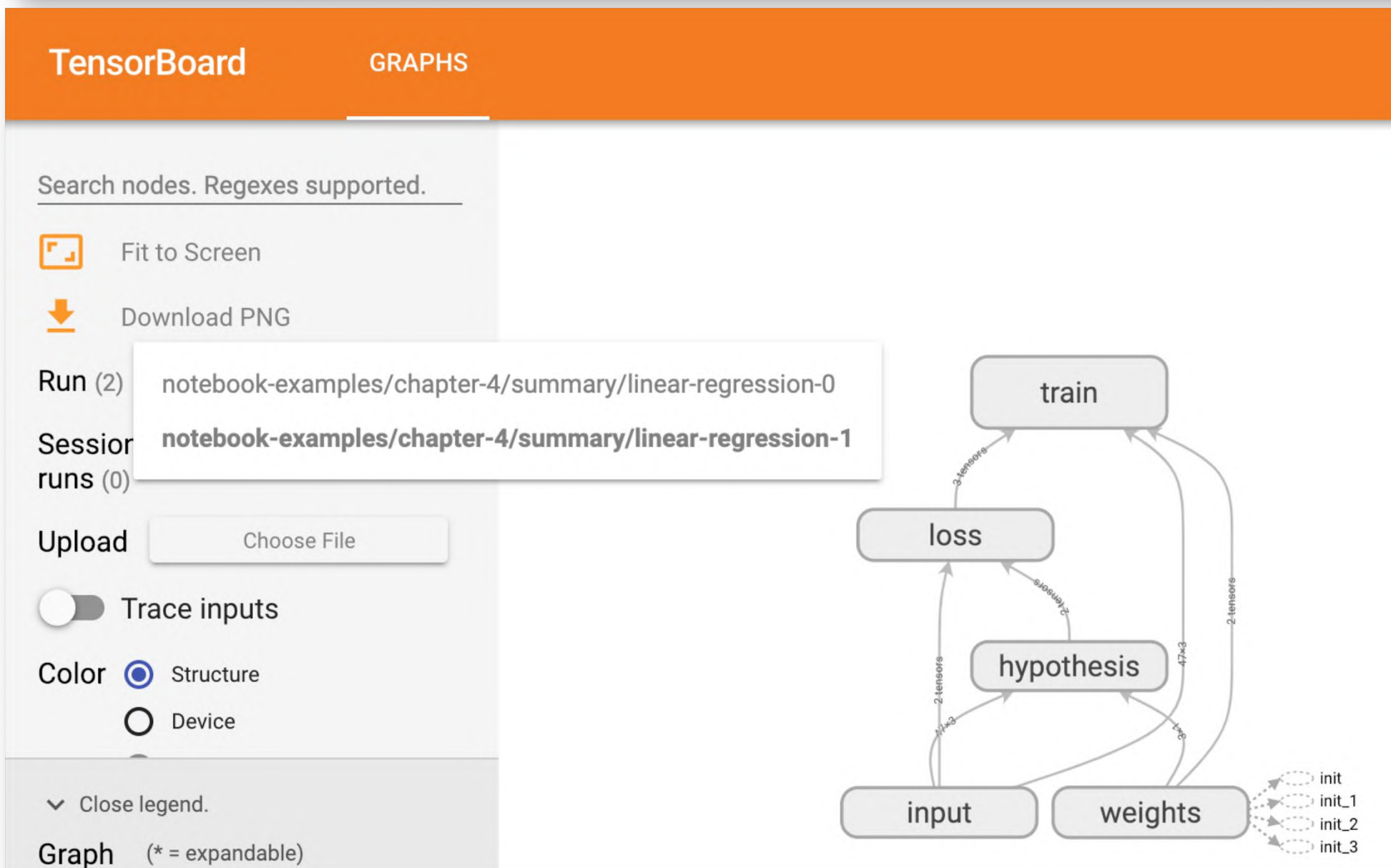
☐ events.out.tfevents.1547832071.Django.lan

15 minutes ago

18.5 kB

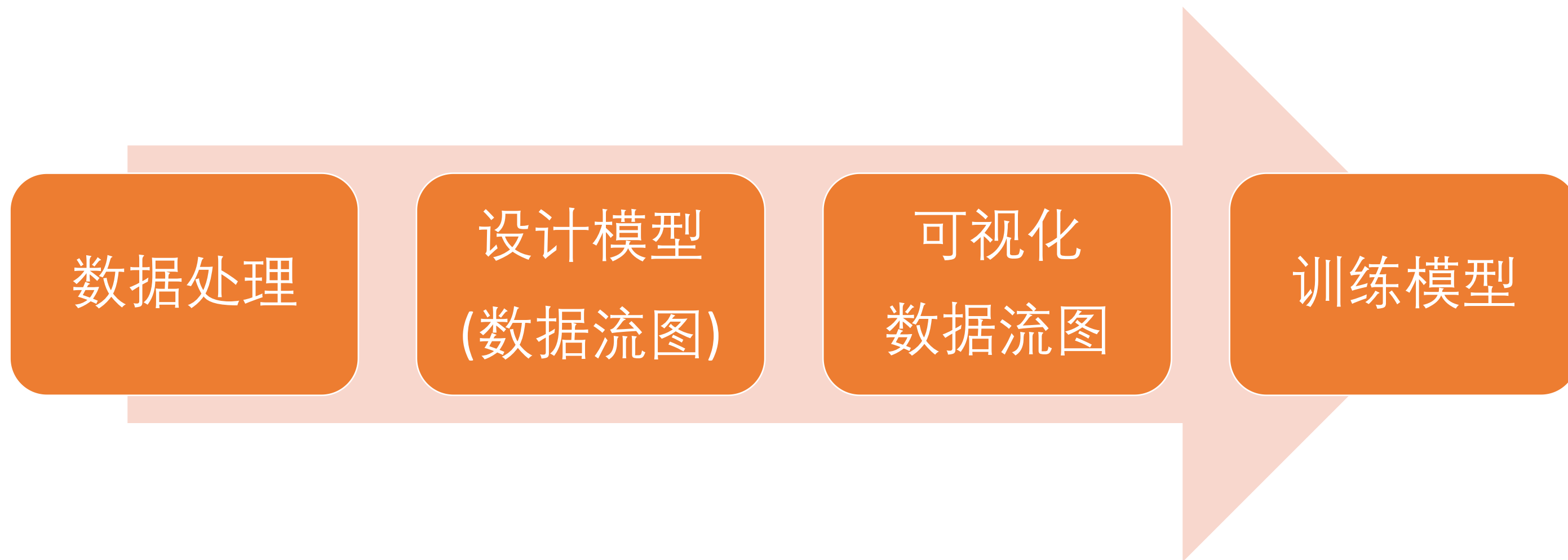
启动 TensorBoard

```
(py36) Django:~/tf-course $ tensorboard --logdir ./ --host localhost
TensorBoard 1.12.2 at http://localhost:6006 (Press CTRL+C to quit)
```



实战 TensorFlow 房价预测

实战 TensorFlow 房价预测 workflow



Try it



扫描二维码

试看/购买 《TensorFlow 快速入门与实战》 视频课程