机器学习入门

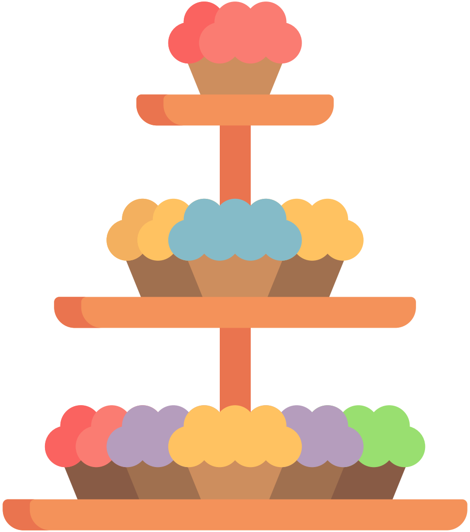
以及如何使用 TensorFlow

2018 年 1 月 25 日

张琦

2012 实验室 · 测试工具部
华为技术有限公司

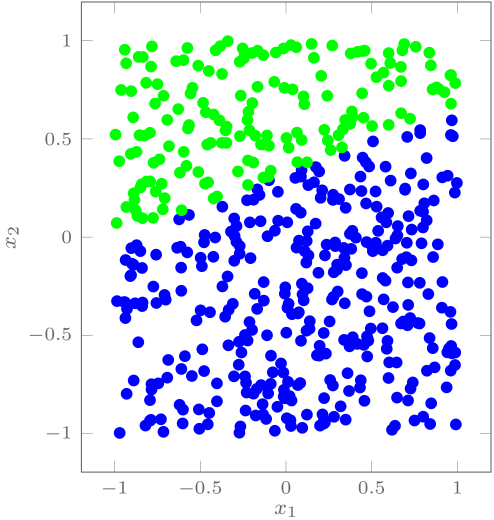
开胃菜



开胃菜
简单的分类问题

在二维空间中，有 500 个点，这 500 个点被分成两类，一类被标记为绿色，一类被标记为蓝色，如右图所示.

500 个点的分布图



开胃菜

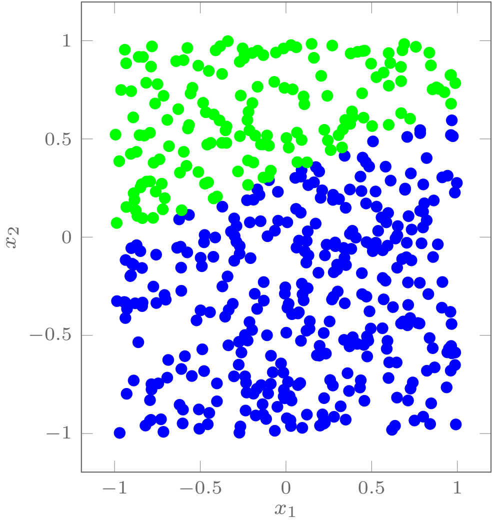
简单的分类问题

在二维空间中，有 500 个点，这 500 个点被分成两类，一类被标记为绿色，一类被标记为蓝色，如右图所示.

现在的问题是，计算机如何根据已有的数据，找到一个分类的方法，可以在给定 (x_1, x_2) 时，会判断点 (x_1, x_2) 的颜色.

计算机解决二分类的方法有很多，在这里，介绍一种基于神经网络的方法，来解决这个问题，来帮助大家理解机器学习的过程.

500 个点的分布图



开胃菜

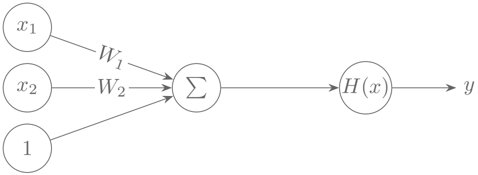
建立神经网络

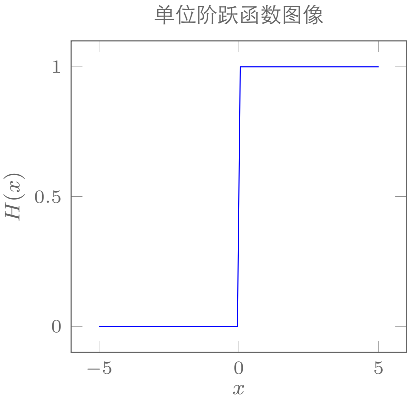
我们不妨设绿色为 1, 蓝色为 0, 那么, 问题就变成了找到一个坐标 (x_1, x_2) 到颜色 y 的一个映射关系 $y = h(x_1, x_2)$.

开胃菜
建立神经网络

我们不妨设绿色为 1, 蓝色为 0, 那么, 问题就变成了找到一个坐标 (x_1, x_2) 到颜色 y 的一个映射关系 $y = h(x_1, x_2)$.

为此, 我们建立一个简单的神经网络, 如图所示.

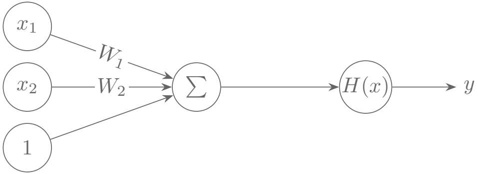


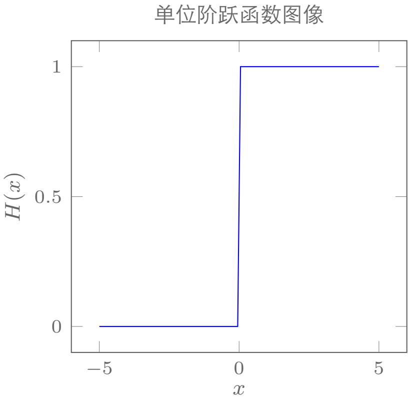


开胃菜
建立神经网络

我们不妨设绿色为 1, 蓝色为 0, 那么, 问题就变成了找到一个坐标 (x_1, x_2) 到颜色 y 的一个映射关系 $y = h(x_1, x_2)$.

为此, 我们建立一个简单的神经网络, 如图所示.





因此, 我们可以得到神经网络的表达式:

$$
\begin{aligned}
y &= H(W_1 x_1 + W_2 x_2 + 1), \\
&= H\big((W_1, W_2) \cdot (x_1, x_2)^\top + 1\big) = H(\boldsymbol{W}\boldsymbol{x} + 1).
\end{aligned}
$$

开胃菜

确定目标函数

所谓目标函数, 就是当目标函数取得最值的时候, 神经网络最符合我们的预期. 换句话就是, 目标函数就是衡量神经网络好坏指标.

开胃菜

确定目标函数

所谓目标函数, 就是当目标函数取得最值的时候, 神经网络最符合我们的预期. 换句话就是, 目标函数就是衡量神经网络好坏指标.

我们这里采用所有样本都分类成功的概率作为目标函数:

$$p(X) = \prod_{i=1}^{500} p(X_i),$$

开胃菜

确定目标函数

所谓目标函数, 就是当目标函数取得最值的时候, 神经网络最符合我们的预期. 换句话就是, 目标函数就是衡量神经网络好坏指标.

我们这里采用所有样本都分类成功的概率作为目标函数:

$$p(X) = \prod_{i=1}^{500} p(X_i),$$

对于第 i 个坐标 $\boldsymbol{x}_i = (x_{i,1}, x_{i,2})$, 其正确值是 y_i, 那么当分类规则 $H(\boldsymbol{W}\boldsymbol{x}+1)$ 给定的情况下, 其作出正确分类的概率为

$$p(X_i) = \begin{cases} 1, & H(\boldsymbol{W}\boldsymbol{x}_i+1) = 1, y_i = 1 \text{ 或者 } H(\boldsymbol{W}\boldsymbol{x}_i+1) = 0, y_i = 0, \\ 0, & H(\boldsymbol{W}\boldsymbol{x}_i+1) = 0, y_i = 1 \text{ 或者 } H(\boldsymbol{W}\boldsymbol{x}_i+1) = 1, y_i = 0. \end{cases}$$

开胃菜
确定目标函数

接下来, 需要对这个式子进行化简.

$$p(X_i) = \begin{cases} 1, & H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 1, y_i = 1 \text{ 或者 } H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 0, y_i = 0, \\ 0, & H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 0, y_i = 1 \text{ 或者 } H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 1, y_i = 0. \end{cases}$$

开胃菜
确定目标函数

接下来，需要对这个式子进行化简.

$$p(X_i) = \begin{cases} 1, & H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 1, y_i = 1 \text{ 或者 } H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 0, y_i = 0, \\ 0, & H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 0, y_i = 1 \text{ 或者 } H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 1, y_i = 0. \end{cases}$$

我们可以注意到，当 $y_i = 1$ 时，$p(X_i)$ 的值与 $H(\boldsymbol{W}\boldsymbol{x}_i + 1)$ 一致；当 $y_i = 0$ 时，$p(X_i)$ 的值与 $H(\boldsymbol{W}\boldsymbol{x}_i + 1)$ 和为 1，于是有:

$$p(X_i) = \begin{cases} H(\boldsymbol{W}\boldsymbol{x}_i + 1), & y_i = 1, \\ 1 - H(\boldsymbol{W}\boldsymbol{x}_i + 1), & y_i = 0. \end{cases}$$

开胃菜
确定目标函数

接下来，需要对这个式子进行化简.

$$p(X_i) = \begin{cases} 1, & H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 1, y_i = 1 \text{ 或者 } H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 0, y_i = 0, \\ 0, & H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 0, y_i = 1 \text{ 或者 } H(\boldsymbol{W}\boldsymbol{x}_i + 1) = 1, y_i = 0. \end{cases}$$

我们可以注意到，当 $y_i = 1$ 时，$p(X_i)$ 的值与 $H(\boldsymbol{W}\boldsymbol{x}_i + 1)$ 一致；当 $y_i = 0$ 时，$p(X_i)$ 的值与 $H(\boldsymbol{W}\boldsymbol{x}_i + 1)$ 和为 1，于是有:

$$p(X_i) = \begin{cases} H(\boldsymbol{W}\boldsymbol{x}_i + 1), & y_i = 1, \\ 1 - H(\boldsymbol{W}\boldsymbol{x}_i + 1), & y_i = 0. \end{cases}$$

注意到 $\forall x \in \mathbb{R}$ 都有 $x^0 = 1$，还可以将上式继续化简为:

$$p(X_i) = H(\boldsymbol{W}\boldsymbol{x}_i + 1)^{y_i} \cdot \big(1 - H(\boldsymbol{W}\boldsymbol{x}_i + 1)\big)^{1-y_i}.$$

开胃菜
确定目标函数

因此, 我们的目标函数就确定下来了:

$$p(X) = \prod_{i=1}^{500} H(\boldsymbol{W}\boldsymbol{x}_i + 1)^{y_i} \cdot \big(1 - H(\boldsymbol{W}\boldsymbol{x}_i + 1)\big)^{1-y_i}.$$

开胃菜
确定目标函数

因此, 我们的目标函数就确定下来了:

$$p(X) = \prod_{i=1}^{500} H(\boldsymbol{W}\boldsymbol{x}_i + 1)^{y_i} \cdot \big(1 - H(\boldsymbol{W}\boldsymbol{x}_i + 1)\big)^{1-y_i}.$$

为了方便计算, 对上式左右两边都取以 e 为底的对数:

$$\ln p(X) = \sum_{i=1}^{500} y_i \ln H(\boldsymbol{W}\boldsymbol{x}_i + 1) + (1 - y_i) \ln \big(1 - H(\boldsymbol{W}\boldsymbol{x}_i + 1)\big).$$

开胃菜
确定目标函数

因此，我们的目标函数就确定下来了：

$$p(X)=\prod_{i=1}^{500} H(\boldsymbol{W}\boldsymbol{x}_i+1)^{y_i}\cdot\big(1-H(\boldsymbol{W}\boldsymbol{x}_i+1)\big)^{1-y_i}.$$

为了方便计算，对上式左右两边都取以 e 为底的对数：

$$\ln p(X)=\sum_{i=1}^{500} y_i \ln H(\boldsymbol{W}\boldsymbol{x}_i+1)+(1-y_i)\ln\big(1-H(\boldsymbol{W}\boldsymbol{x}_i+1)\big).$$

但是，上述目标函数还存在问题：

- 只要有一个预测错误，函数 $\ln p(X)=-\infty$，目标函数并不能衡量模型的精度.
- 导致函数 $\ln p(X)$ 不可微，不能采用梯度下降法来计算函数的最值 (其实是极值);

开胃菜

确定目标函数

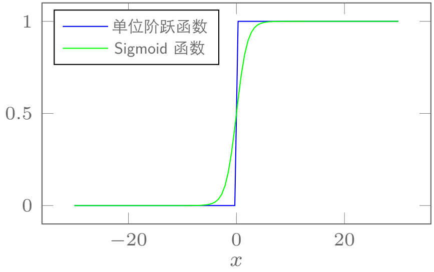
解决方法就是用 Sigmoid 函数 $g(\cdot)$ 代替上式中的单位阶跃函数 $H(\cdot)$, 原因有:

开胃菜
确定目标函数

解决方法就是用 Sigmoid 函数 $g(\cdot)$ 代替上式中的单位阶跃函数 $H(\cdot)$, 原因有:

- Sigmoid 函数与单位阶跃函数形状非常相似, 并且值域为 $(0, 1)$, 不会导致目标函数出现 $-\infty$ 的情况, 其函数图像如右图所示.

Sigmoid 函数与单位阶跃函数图像



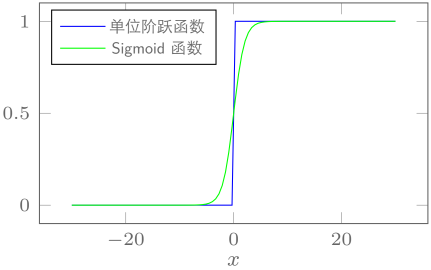
开胃菜
确定目标函数

解决方法就是用 Sigmoid 函数 $g(\cdot)$ 代替上式中的单位阶跃函数 $H(\cdot)$，原因有:

- Sigmoid 函数与单位阶跃函数形状非常相似，并且值域为 $(0,1)$，不会导致目标函数出现 $-\infty$ 的情况，其函数图像如右图所示.
- Sigmoid 处处可微，Sigmoid 的表达式为:

$$g(x) = \frac{1}{1+\exp(-x)}.$$

Sigmoid 函数与单位阶跃函数图像



开胃菜
确定目标函数

解决方法就是用 Sigmoid 函数 $g(\cdot)$ 代替上式中的单位阶跃函数 $H(\cdot)$, 原因有:

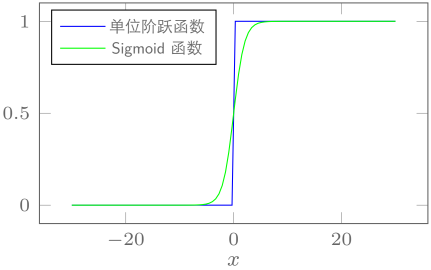
- Sigmoid 函数与单位阶跃函数形状非常相似, 并且值域为 $(0,1)$, 不会导致目标函数出现 $-\infty$ 的情况, 其函数图像如右图所示.
- Sigmoid 处处可微, Sigmoid 的表达式为:

$$g(x) = \frac{1}{1+\exp(-x)}.$$

- Sigmoid 函数有非常好的导函数性质, 有:

$$\frac{\mathrm{d}g(x)}{\mathrm{d}x} = g(x)\cdot(1-g(x)).$$

Sigmoid 函数与单位阶跃函数图像



开胃菜
确定目标函数

解决方法就是用 Sigmoid 函数 $g(\cdot)$ 代替上式中的单位阶跃函数 $H(\cdot)$, 原因有:

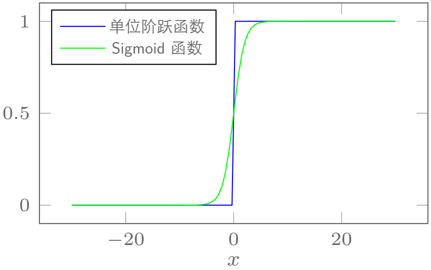
- Sigmoid 函数与单位阶跃函数形状非常相似, 并且值域为 $(0,1)$, 不会导致目标函数出现 $-\infty$ 的情况, 其函数图像如右图所示.
- Sigmoid 处处可微, Sigmoid 的表达式为:

$$g(x) = \frac{1}{1+\exp(-x)}.$$

- Sigmoid 函数有非常好的导函数性质, 有:

$$\frac{\mathrm{d}g(x)}{\mathrm{d}x} = g(x)\cdot\big(1-g(x)\big).$$

Sigmoid 函数与单位阶跃函数图像



目标函数: $$\ell(\boldsymbol{W}) = -\sum_{i=1}^{500} y_i \ln g(\boldsymbol{W}\boldsymbol{x}_i+1) + (1-y_i)\ln\big(1-g(\boldsymbol{W}\boldsymbol{x}_i+1)\big).$$

开胃菜

参数训练

参数训练的本质是找到一个参数 $\boldsymbol{W}$ 使目标函数 $\ell(\boldsymbol{W})$ 取得最小值, 常用的是梯度下降法.

开胃菜

参数训练

参数训练的本质是找到一个参数 $\boldsymbol{W}$ 使目标函数 $\ell(\boldsymbol{W})$ 取得最小值, 常用的是梯度下降法.

把 $\boldsymbol{W} = (W_1, W_2)$ 看成是平面坐标, $\ell(\boldsymbol{W})$ 看作是坐标 (W_1, W_2) 的海拔高度, 梯度下降法的策略就是每次向最陡的方向走一步, 一直走到海拔不变为止, 此时就是海拔 “最低点”, 即 $\ell(\boldsymbol{W})$ 的极小值点.

开胃菜

参数训练

参数训练的本质是找到一个参数 $\boldsymbol{W}$ 使目标函数 $\ell(\boldsymbol{W})$ 取得最小值, 常用的是梯度下降法.

把 $\boldsymbol{W} = (W_1, W_2)$ 看成是平面坐标, $\ell(\boldsymbol{W})$ 看作是坐标 (W_1, W_2) 的海拔高度, 梯度下降法的策略就是每次向最陡的方向走一步, 一直走到海拔不变为止, 此时就是海拔 “最低点”, 即 $\ell(\boldsymbol{W})$ 的极小值点.

梯度下降的迭代公式为:

$$\boxed{\boldsymbol{W}_n} \leftarrow \boxed{\boldsymbol{W}_{n-1}} - \boxed{\boldsymbol{r}} \cdot \boxed{\left.\frac{\partial \ell(\boldsymbol{W})}{\partial \boldsymbol{W}}\right|_{\boldsymbol{W}=\boldsymbol{W}_{n-1}}},$$

其中:

- 下一步的位置;
- 当前站的位置;
- 步长, 也被称为学习率;
- 最陡的方向, 也就是梯度.

开胃菜

参数训练

我们计算一下 $\frac{\partial \ell(\boldsymbol{W})}{\partial \boldsymbol{W}}$, 设 $z = \boldsymbol{W}\boldsymbol{x} + 1 = W_1x_1 + W_2x_2 + 1$, 于是:

开胃菜

参数训练

我们计算一下 $\frac{\partial \ell(\boldsymbol{W})}{\partial \boldsymbol{W}}$, 设 $z = \boldsymbol{W}\boldsymbol{x} + 1 = W_1 x_1 + W_2 x_2 + 1$, 于是:

$$\frac{\partial \ell(\boldsymbol{W})}{\partial z} = -\frac{\partial \sum_{i=1}^{500} \Big(y_i \ln g(z) + (1 - y_i) \ln \big(1 - g(z)\big) \Big)}{\partial z},$$

开胃菜
参数训练

我们计算一下 $\frac{\partial \ell(\boldsymbol{W})}{\partial \boldsymbol{W}}$，设 $z = \boldsymbol{W}\boldsymbol{x} + 1 = W_1 x_1 + W_2 x_2 + 1$，于是:

$$\begin{aligned}\frac{\partial \ell(\boldsymbol{W})}{\partial z} &= -\frac{\partial \sum_{i=1}^{500}\Big(y_i \ln g(z) + (1-y_i)\ln\big(1-g(z)\big)\Big)}{\partial z}, \\ &= -\sum_{i=1}^{500} y_i \frac{\partial \ln g(z)}{\partial z} + (1-y_i)\frac{\partial \ln\big(1-g(z)\big)}{\partial z} = -\sum_{i=1}^{500} y_i \frac{g'(z)}{g(z)} + (1-y_i)\frac{-g'(z)}{1-g(z)},\end{aligned}$$

开胃菜

参数训练

我们计算一下 $\frac{\partial \ell(\boldsymbol{W})}{\partial \boldsymbol{W}}$, 设 $z = \boldsymbol{W}\boldsymbol{x} + 1 = W_1 x_1 + W_2 x_2 + 1$, 于是:

$$
\begin{aligned}
\frac{\partial \ell(\boldsymbol{W})}{\partial z} &= -\frac{\partial \sum_{i=1}^{500}\Big(y_i \ln g(z) + (1-y_i)\ln\big(1-g(z)\big)\Big)}{\partial z}, \\
&= -\sum_{i=1}^{500} y_i \frac{\partial \ln g(z)}{\partial z} + (1-y_i)\frac{\partial \ln\big(1-g(z)\big)}{\partial z} = -\sum_{i=1}^{500} y_i \frac{g'(z)}{g(z)} + (1-y_i)\frac{-g'(z)}{1-g(z)}, \\
&= -\sum_{i=1}^{500} y_i \frac{g(z)(1-g(z))}{g(z)} + (1-y_i)\frac{-g(z)(1-g(z))}{1-g(z)} = -\sum_{i=1}^{500} y_i - g(z).
\end{aligned}
$$

开胃菜

参数训练

我们计算一下 $\frac{\partial \ell(\boldsymbol{W})}{\partial \boldsymbol{W}}$，设 $z = \boldsymbol{W}\boldsymbol{x} + 1 = W_1 x_1 + W_2 x_2 + 1$，于是:

$$
\begin{aligned}
\frac{\partial \ell(\boldsymbol{W})}{\partial z} &= -\frac{\partial \sum_{i=1}^{500} \Big(y_i \ln g(z) + (1-y_i) \ln \big(1 - g(z)\big) \Big)}{\partial z}, \\
&= -\sum_{i=1}^{500} y_i \frac{\partial \ln g(z)}{\partial z} + (1-y_i) \frac{\partial \ln \big(1-g(z)\big)}{\partial z} = -\sum_{i=1}^{500} y_i \frac{g'(z)}{g(z)} + (1-y_i)\frac{-g'(z)}{1-g(z)}, \\
&= -\sum_{i=1}^{500} y_i \frac{g(z)(1-g(z))}{g(z)} + (1-y_i)\frac{-g(z)(1-g(z))}{1-g(z)} = -\sum_{i=1}^{500} y_i - g(z).
\end{aligned}
$$

$$
\frac{\partial \ell(\boldsymbol{W})}{\partial W_1} = \frac{\partial \ell(\boldsymbol{W})}{\partial z} \cdot \frac{\partial z}{\partial W_1} = -\sum_{i=1}^{500} \big(y_i - g(W_1 x_{i,1} + W_2 x_{i,2} + 1)\big) \cdot x_{i,1},
$$

$$
\frac{\partial \ell(\boldsymbol{W})}{\partial W_2} = \frac{\partial \ell(\boldsymbol{W})}{\partial z} \cdot \frac{\partial z}{\partial W_2} = -\sum_{i=1}^{500} \big(y_i - g(W_1 x_{i,1} + W_2 x_{i,2} + 1)\big) \cdot x_{i,2}.
$$

开胃菜
代码实现


```python
from math import exp, log

def sigmoid(x, n = 1): # 定义 Sigmoid 函数 $g = 1/(1 + \exp(x))$
  return 1 / (1 + exp(-n * x)) if x >= 0 else exp(n * x) / (1 + exp(n * x))

def z(x1, x2, w1, w2): # 定义中间变量 $z = W_1 x_1 + W_2 x_2 + 1$
  return w1 * x1 + w2 * x2 + 1

def l(x1, x2, y, w1, w2): # 定义目标函数 $\ell(\boldsymbol{W}) = -\sum_{i=1}^{500} y_i \ln g(\boldsymbol{W}\boldsymbol{x}_i + 1) + (1 - y_i) \ln \left(1 - g(\boldsymbol{W}\boldsymbol{x}_i + 1)\right)$
  return -sum([y * log(sigmoid(z(x1, x2, w1, w2))) + (1 - y) * log(1 - sigmoid(z(x1, x2, w1,
    ↪  w2))) for x1, x2, y in zip(x1, x2, y)])

def get_delta_w1(x1, x2, y, w1, w2): # 定义 $\frac{\partial \ell(\boldsymbol{W})}{\partial W_1} = -\sum_{i=1}^{500} \left(y_i - g(W_1 x_{i,1} + W_2 x_{i,2} + 1)\right) \cdot x_{i,1}$
  return -sum([(y - sigmoid(z(x1, x2, w1, w2))) * x1 for x1, x2, y in zip(x1, x2, y)])

def get_delta_w2(x1, x2, y, w1, w2): # 定义 $\frac{\partial \ell(\boldsymbol{W})}{\partial W_2} = -\sum_{i=1}^{500} \left(y_i - g(W_1 x_{i,1} + W_2 x_{i,2} + 1)\right) \cdot x_{i,2}$
  return -sum([(y - sigmoid(z(x1, x2, w1, w2))) * x2 for x1, x2, y in zip(x1, x2, y)])
```

开胃菜

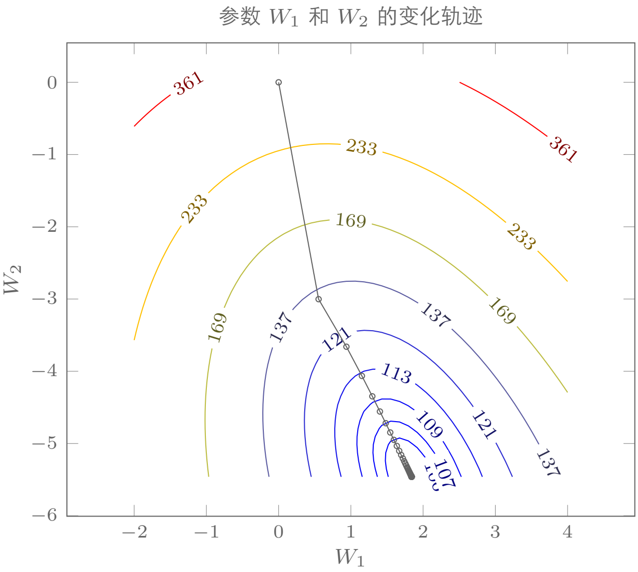
代码实现

```python
with open('training.dat', 'r') as data_file: # 读取训练数据 x1, x2 和 y
  next(data_file)
  data = zip(*[[float(x.strip()) for x in line.split(',')] for line in data_file if not line.strip() == ''])
x1, x2, y = (column for column in data)

w1, w2 = 0, 0 # 初始化变量 W1 和 W2
r = 0.03 # 设置学习率
precision = 1e-4 # 设置训练的目标精度

while True: # 不断的进行循环迭代，直至 W1 和 W2 的值都不变为止
  delta_w1, delta_w2 = get_delta_w1(x1, x2, y, w1, w2), get_delta_w2(x1, x2, y, w1, w2)
  w1, w2 = w1 - r * delta_w1, w2 - r * delta_w2
  if delta_w1**2 + delta_w2**2 < precision**2:
    break

print('w1 = {w1}, w2 = {w2}'.format(w1 = w1, w2 = w2)) # 打印结果
```

开胃菜

代码实现

在训练的过程中，我们将每一次迭代的结果记录下来，并绘制成曲线图，如右图所示.

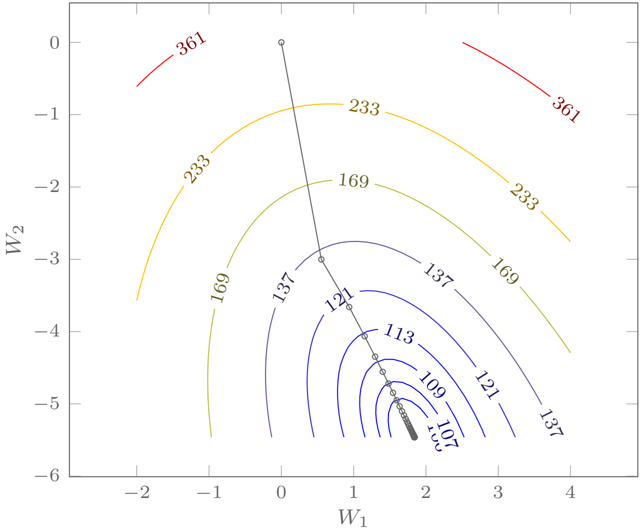


开胃菜
代码实现

在训练的过程中，我们将每一次迭代的结果记录下来，并绘制成曲线图，如右图所示.

- 每一步的方向都是垂直于等高线，即梯度的方向，这就是梯度下降法的由来;

参数 W_1 和 W_2 的变化轨迹



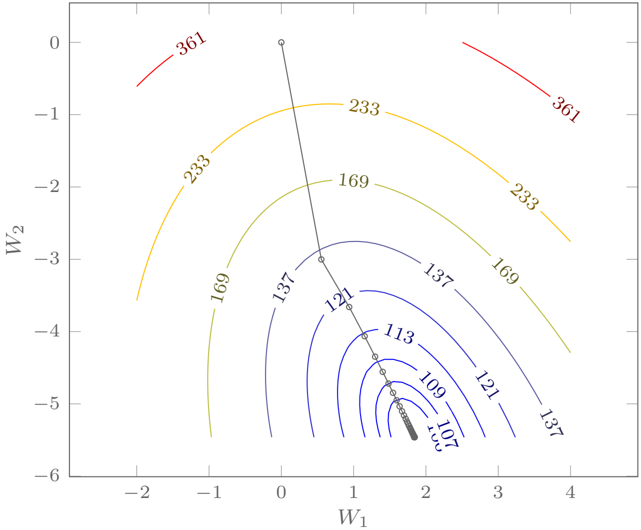
开胃菜

代码实现

在训练的过程中，我们将每一次迭代的结果记录下来，并绘制成曲线图，如右图所示.

- 每一步的方向都是垂直于等高线，即梯度的方向，这就是梯度下降法的由来;
- 步长越来越小，这是因为坡度越来越缓;

参数 W_1 和 W_2 的变化轨迹



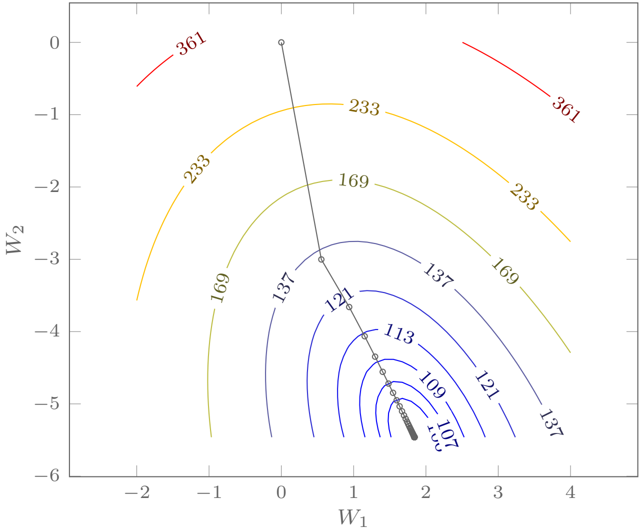
开胃菜

代码实现

在训练的过程中，我们将每一次迭代的结果记录下来，并绘制成曲线图，如右图所示.

- 每一步的方向都是垂直于等高线，即梯度的方向，这就是梯度下降法的由来;
- 步长越来越小，这是因为坡度越来越缓;
- 最终参数 W_1 和 W_2 收敛于 $(1.84, -5.46)$.

参数 W_1 和 W_2 的变化轨迹



开胃菜

结果分析

由于参数 W_1 和 W_2 收敛于 $(1.84, -5.46)$, 我们可以得到一条直线:

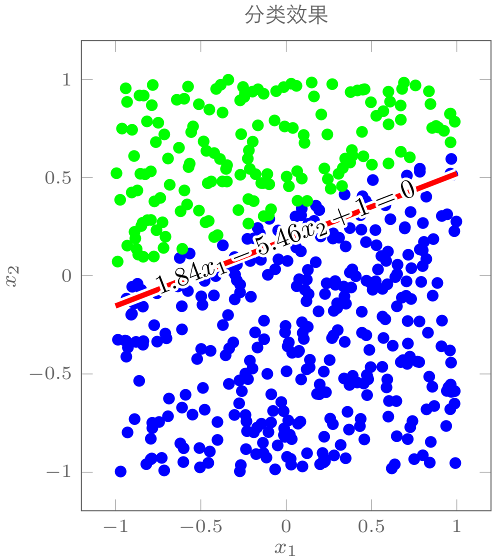
$$1.84x_1 - 5.46x_2 + 1 = 0.$$

开胃菜
结果分析

由于参数 W_1 和 W_2 收敛于 $(1.84, -5.46)$, 我们可以得到一条直线:

$$1.84x_1 - 5.46x_2 + 1 = 0.$$

如右图所示, 可见, 结果很不理想. 通过分析数据, 我们可以看出, 点 $(1.84, -5.46)$ 处的目标函数值为 $\ell(1.84, -5.46) = 105.11$, 也是非常不理想的.



开胃菜
结果分析

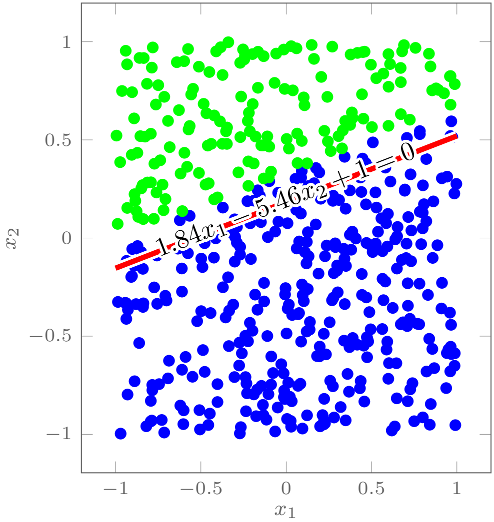
由于参数 W_1 和 W_2 收敛于 $(1.84, -5.46)$, 我们可以得到一条直线:

$$1.84x_1 - 5.46x_2 + 1 = 0.$$

如右图所示, 可见, 结果很不理想. 通过分析数据, 我们可以看出, 点 $(1.84, -5.46)$ 处的目标函数值为 $\ell(1.84, -5.46) = 105.11$, 也是非常不理想的.

这是由于用 Sigmoid 函数代替单位阶跃函数时, 给目标函数带来的误差导致的.

分类效果



开胃菜
改进激活函数

我们对 Sigmoid 函数表达式做稍加改进, 如下所示.

$$g(x, n) = \frac{1}{1 + \exp(n \cdot x)}.$$

开胃菜
改进激活函数

我们对 Sigmoid 函数表达式做稍加改进，如下所示.

$$g(x, n) = \frac{1}{1 + \exp(n \cdot x)}.$$

分别绘制单位阶跃函数，$g(x, 1)$ 和 $g(x, 5)$，可以看出，随着 n 的增大，函数 $g(x, n)$ 越来越接近于单位阶跃函数 $H(x)$.

开胃菜

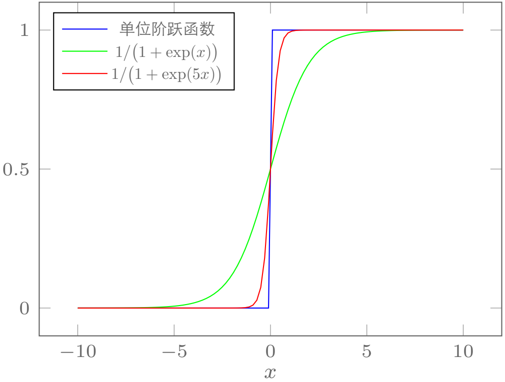
改进激活函数

我们对 Sigmoid 函数表达式做稍加改进，如下所示.

$$g(x,n) = \frac{1}{1+\exp(n\cdot x)}.$$

分别绘制单位阶跃函数，$g(x,1)$ 和 $g(x,5)$，可以看出，随着 n 的增大，函数 $g(x,n)$ 越来越接近于单位阶跃函数 $H(x)$.

Sigmoid 函数与单位阶跃函数图像



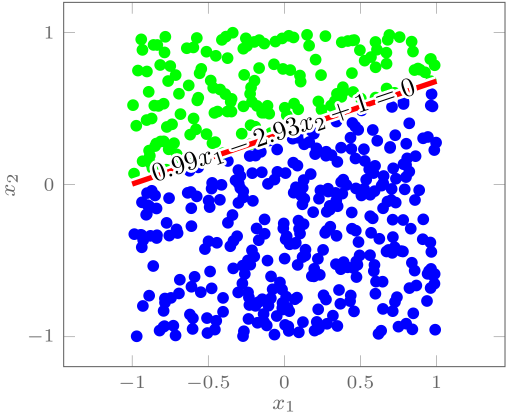
并且可以证明: $\forall x\in\mathbb{R},\quad \lim_{n\to+\infty} g(x,n) = \lim_{n\to+\infty}\frac{1}{1+\exp(n\cdot x)} = H(x).$

开胃菜

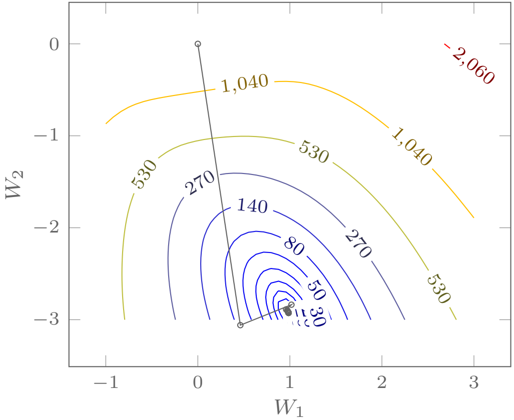
改进激活函数

令 $n = 10$, 参数 W_1 和 W_2 收敛于 $(0.99, -2.93)$. 分类效果和参数 W_1 和 W_2 轨迹如下所示.

分类效果



参数 W_1 和 W_2 的变化轨迹

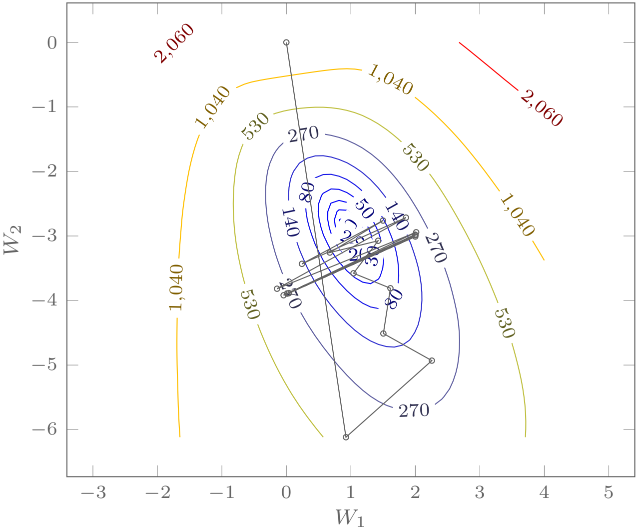


开胃菜

学习率

我们把学习率 r 提高，重新进行学习，参数 W_1 和 W_2 的变化轨迹如右图所示.

参数 W_1 和 W_2 的变化轨迹



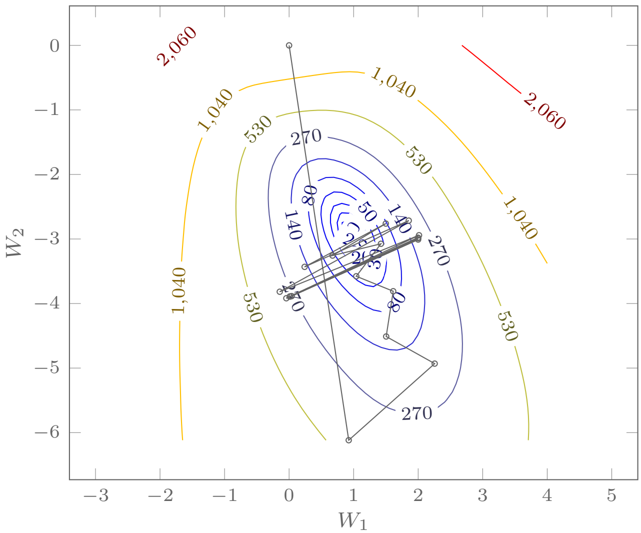
开胃菜

学习率

我们把学习率 r 提高，重新进行学习，参数 W_1 和 W_2 的变化轨迹如右图所示.

可以看到，整个目标函数的最优点仍然是 $(0.99, -2.93)$，但是，参数 W_1 和 W_2 无法收敛于最优点.

参数 W_1 和 W_2 的变化轨迹



开胃菜

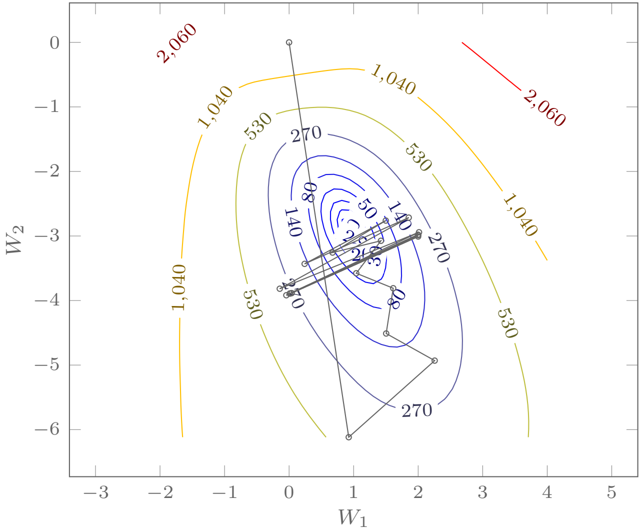
学习率

我们把学习率 r 提高，重新进行学习，参数 W_1 和 W_2 的变化轨迹如右图所示.

可以看到，整个目标函数的最优点仍然是 $(0.99, -2.93)$，但是，参数 W_1 和 W_2 无法收敛于最优点.

相对的，过小的学习率会导致特学习过程非常缓慢. 故选择合适大小的学习率是非常重要的.

参数 W_1 和 W_2 的变化轨迹



开胃菜
总结

让我们总结一下这一小节关键点:

- 机器学习的本质是重现人认识世界的过程, 具体实现靠的是空间搜索和函数的泛化;
- 机器学习的一般过程:
  - 数据分析以及预处理;
  - 选择合适的模型;
  - 设定目标函数以及训练参数;
  - 验证以及调整模型.

开胃菜
总结

让我们总结一下这一小节关键点:

- 机器学习的本质是重现人认识世界的过程, 具体实现靠的是空间搜索和函数的泛化;
- 机器学习的一般过程:
  - 数据分析以及预处理;
  - 选择合适的模型;
  - 设定目标函数以及训练参数;
  - 验证以及调整模型.

但是, 机器学习的过程复杂, 需要计算非常复杂的目标函数, 梯度的表达式, 编码过程中非常容易出错, 有大量的多维数据运算, 计算缓慢……

TensorFlow 简易教程

TensorFlow 简易教程

TensorFlow 简介

TensorFlow 是一款通过数据流图进行数值计算的开源库. TensorFlow 最早是被 Google Brain 的研究者和工程师开发出来的用于机器学习和深度神经网络的研究. 但是这个系统也同样适用于其他领域.

TensorFlow 具有很多非常好的特性:

- **易用性**, TensorFlow 封装了很多复杂运算, 使用者只需构建计算图.
- **可移植性**, TensorFlow 支持多种平台, 系统以及硬件.
- **自动求导**, TensorFlow 支持自动求导, 给采用基于梯度下降的学习方法带来了极大便利.
- **支持多种语言**, TensorFlow 提供易用的 Python 接口, 也支持 C++, Java, Go 等语言.
- **性能最大化**, 可以部署到不同硬件, 并使用线程, 队列, 异步计算来最大化硬件使用效率.

TensorFlow 简易教程

TensorFlow 简介

编程模型	Dataflow-Like Model
语言	Python, C++, Go, Rust, Haskell, Java, Julia, JavaScript, R
部署	Code-once, run everywhere
计算资源	CPU, GPU, TPU
实现方式	Local Implementation, Distributed Implementation
平台支持	Google Cloud Platform, Hadoop File System
数学表达	Math Graph Expression, Auto Differentiation
优化	Common Subexpression Elimination, Asynchronous Kernel Optimization, Communication Optimization, Model Parallelism, Data Parallelism, Pipeline

TensorFlow 简易教程

张量 Tensor

张量最初是个物理学的概念:

> A tensor is something that transforms like a tensor
>
> 一个量, 在不同的参考系下按照某种特定的法则进行变换, 就是张量.
>
> — A. Zee, "Einstein Gravity in a Nutshell"

TensorFlow 简易教程

张量 Tensor

张量最初是个物理学的概念:

> A tensor is something that transforms like a tensor
>
> 一个量, 在不同的参考系下按照某种特定的法则进行变换, 就是张量.
>
> — A. Zee, "Einstein Gravity in a Nutshell"

在 TensorFlow 中, 可以将张量理解成多维数组, 张量穿梭在数据流图中, 当经过节点时, 会被节点按照特定法则变换成另一个张量, 这个就是 TensorFlow 名字的由来.

TensorFlow 简易教程

张量 Tensor

张量最初是个物理学的概念:

> A tensor is something that transforms like a tensor
>
> 一个量, 在不同的参考系下按照某种特定的法则进行变换, 就是张量.
>
> — A. Zee, "Einstein Gravity in a Nutshell"

在 TensorFlow 中, 可以将张量理解成多维数组, 张量穿梭在数据流图中, 当经过节点时, 会被节点按照特定法则变换成另一个张量, 这个就是 TensorFlow 名字的由来.

张量的维数被称为张量的阶, 我们常见的标量, 其实是 0 阶张量, 矢量就是 1 阶张量, 矩阵就是 2 阶张量, 一幅图片, 在 TensorFlow 中可以是 3 阶张量.

TensorFlow 简易教程

张量 Tensor

TensorFlow 文档中使用了三种记号来方便地描述张量的维度：阶，形状以及维数．下表展示了他们之间的关系：

阶	形状	维数	python 代码
0	[ ]	0	`t0 = 483`
1	$[D_0]$	1	`t1 = [1.1, 2.2, 3.3]`
2	$[D_0, D_1]$	2	`t2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
3	$[D_0, D_1, D_2]$	3	`t3 = [[[1], [2]], [[3], [4]], [[5], [6]]]`
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	$[D_0, D_1, \cdots, D_n]$	n	`tn = ...`

TensorFlow 简易教程

张量 Tensor

除了维度，Tensors 有一个数据类型属性．可为一个张量指定下列数据类型的任意一个类型：

类型	描述	类型	描述
`tf.float32`	32 位浮点数	`tf.string`	可变长度的字节数组
`tf.float64`	64 位浮点数	`tf.bool`	布尔型
`tf.int64`	64 位有符号整型	`tf.complex64`	由两个 32 位浮点数组成的复数
`tf.int32`	32 位有符号整型	`tf.qint32`	用于量化操作的 32 位有符号整型
`tf.int16`	16 位有符号整型	`tf.qint8`	用于量化操作的 8 位有符号整型
`tf.int8`	8 位有符号整型	`tf.quint8`	用于量化操作的 8 位无符号整型
`tf.uint8`	8 位无符号整型		

TensorFlow 简易教程

张量 Tensor

TensorFlow 中张量包括 `constant`, `placeholder` 和 `Variable`.

TensorFlow 简易教程

张量 Tensor

TensorFlow 中张量包括 `constant`, `placeholder` 和 `Variable`.

`tf.constant()` 函数提供在 TensorFlow 中定义不可更改张量的方法, 定义如下所示.

```python
def constant(value, dtype=None, shape=None, name="Const", verify_shape=False)
```

- `value`, 符合 TensorFlow 中定义的数据类型的常数值或者常数列表;
- `dtype`, 数据类型, 可选;
- `shape`, 常量的形状, 可选;
- `name`, 常量的名字, 可选;
- `verify_shape`, 常量的形状是否可以被更改, 默认不可更改.

除了直接赋值以外, 还可使用 `tf.ones()`, `tf.zeros()` 等初始化张量的方法.

TensorFlow 简易教程

张量 Tensor

TensorFlow 中张量包括 `constant`, `placeholder` 和 `Variable`.

`tf.placeholder()` 函数提供在 TensorFlow 中定义占位张量的方法, 定义如下所示.

```
def placeholder(dtype, shape=None, name=None)
```

- `dtype` 指定占位张量的数据类型, 可以是 TensorFlow 中的数据类型, 如常用的 `tf.float32`, `tf.float64` 等数值类型;
- `shape` 表示数据类型, `shape = [None, 5]`, 表示行不定, 列是 `5`;
- `name` 是张量名称;

占位变量是一种 TensorFlow 用来解决读取大量训练数据问题的机制, 它允许你现在不用给它赋值, 随着训练的开始, 再把训练数据传送给训练网络学习.

TensorFlow 简易教程

张量 Tensor

TensorFlow 中张量包括 `constant`, `placeholder` 和 `Variable`.

TensorFlow 中变量是通过 `Variable` 类来实现的, 初始化函数定义如下.

```
def __init__(self, initial_value=None, trainable=True, collections=None,
    validate_shape=True, caching_device=None, name=None, variable_def=None,
    dtype=None, expected_shape=None, import_scope=None)
```

- `initial_value`, 初始值, 必填, 张量或可以转换为张量的 Python 对象;
- `trainable`, 如果参数 `trainable` 的值为 `True`, 则默认值也将变量添加到图形中集合 `GraphKeys.TRAINABLE_VARIABLES`. 这个集合用作 `Optimizer` 类使用的默认变量列表;
- `collections`, 新的变量被添加到这些集合. 默认为 `[GraphKeys.GLOBAL_VARIABLES]`;

TensorFlow 简易教程

张量 Tensor

TensorFlow 中张量包括 `constant`, `placeholder` 和 `Variable`.

TensorFlow 中变量是通过 `Variable` 类来实现的, 初始化函数定义如下.

```
def __init__(self, initial_value=None, trainable=True, collections=None,
    validate_shape=True, caching_device=None, name=None, variable_def=None,
    dtype=None, expected_shape=None, import_scope=None)
```

- `validate_shape`, 如果 `False`, 允许变量用初始化未知形状的值. 如果 `True`, 默认的形状 `initial_value` 必须是已知的;
- `caching_device`, 指定变量的默认缓存位置;
- `name`, 是张量名称;

TensorFlow 简易教程

张量 Tensor

TensorFlow 中张量包括 `constant`, `placeholder` 和 `Variable`.

TensorFlow 中变量是通过 `Variable` 类来实现的, 初始化函数定义如下.

```python
def __init__(self, initial_value=None, trainable=True, collections=None,
    validate_shape=True, caching_device=None, name=None, variable_def=None,
    dtype=None, expected_shape=None, import_scope=None)
```

- `variable_def`, 指定 `VariableDef` 的协议缓冲区;
- `dtype`, 指定张量的数据类型;
- `expected_shape`, 是张量的形状, 如果设置, `initial_value` 需要符合这个形状;
- `import_scope`, 张量名称前缀.

TensorFlow 简易教程

节点 Operator

在 TensorFlow 中，每一个节点代表着一个操作，一般用来表示施加的的数学运算，也可以表示数据输入的起点以及输出的终点．下面是一些重要的操作:

操作	描述	操作	描述
`tf.add(x, y, name=None)`	求和	`tf.sign(x, name=None)`	返回符号
`tf.sub(x, y, name=None)`	减法	`tf.neg(x, name=None)`	取负 $(y = -x)$
`tf.mul(x, y, name=None)`	乘法	`tf.square(x, name=None)`	计算平方 $(y = x^2)$
`tf.div(x, y, name=None)`	除法	`tf.round(x, name=None)`	求最接近的整数
`tf.mod(x, y, name=None)`	取模	`tf.sqrt(x, name=None)`	开根号 $(y = \sqrt{x})$
`tf.pow(x, y, name=None)`	乘幂	`tf.abs(x, name=None)`	绝对值
`tf.inv(x, name=None)`	取反	`tf.exp(x, name=None)`	计算 e 的次方

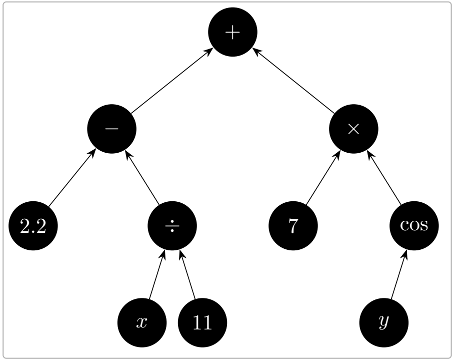
TensorFlow 简易教程

数据流图 Graph

TensorFlow 是用数据流图对计算过程进行描述的. 在数据流图中, 节点代表数学运算, 边表示节点之间的某种联系, 负责在节点之间传输即张量.

节点可以被分配到多个计算设备上, 可以异步和并行的进行操作. 因为是有向图, 所以只能等待之前的节点运行结束, 当前节点才能执行操作.

如图所示是一个简单的数据流图.

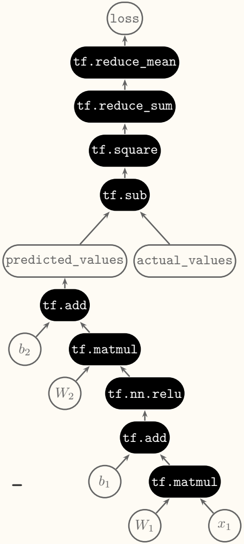


表达式: $\left(2.2-\left(\frac{x}{11}\right)\right)+(7\times\cos(y))$.

TensorFlow 简易教程

数据流图 Graph

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

x1 = tf.placeholder(tf.float32, [None, 1])
w1 = tf.Variable(tf.random_normal([1, 10]))
b1 = tf.Variable(tf.ones([1, 10]))

x2 = tf.nn.relu(tf.matmul(x1, w1) + b1)
w2 = tf.Variable(tf.random_normal([10, 1]))
b2 = tf.Variable(tf.ones([1, 1]))

predicted_values = tf.matmul(x2, w2) + b2
actual_values = tf.placeholder(tf.float32, [None, 1])
loss = tf.reduce_mean(tf.reduce_sum(tf.square(actual_values - predict_values), reduction_indices = [1]))
```



TensorFlow 简易教程

会话 Session

构造阶段完成后，才能启动图．启动图的第一步是创建一个 Session 对象，如果无任何创建参数，会话构造器将启动默认图．会话会管理 TensorFlow 程序运行时的所有资源．当所有计算完成之后需要关闭会话来帮助系统回收资源，否则就可能出现资源泄露的问题．

```python
import tensorflow as tf

a = tf.constant([1.0, 2.0])
b = tf.constant([2.0, 3.0])
result = a + b

session = tf.Session()
print(session.run(result))
session.close()
```

```python
import tensorflow as tf

a = tf.constant([1.0, 2.0])
b = tf.constant([2.0, 3.0])
result = a + b

with tf.Session() as session:
    print(session.run(result))
# session.close()
```

TensorFlow 实战

3.1 分类问题

3.2 拟合问题

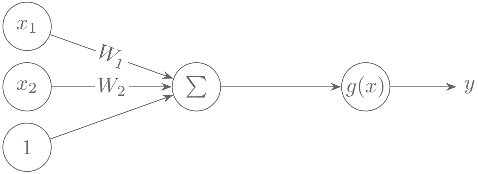
3.3 调试

TensorFlow 实战

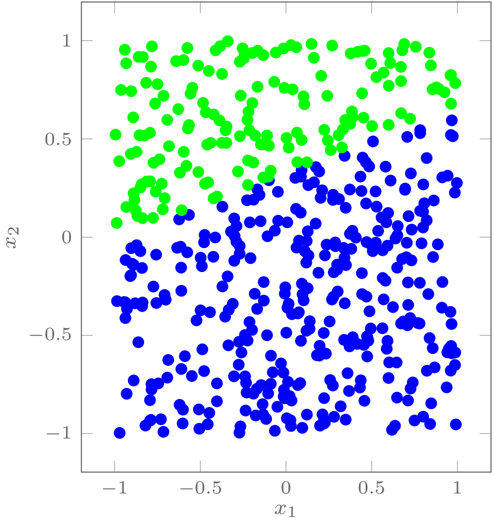
分类问题

回到最开始的那个二分类问题，看看用 TensorFlow 是如何解决这个问题的，500 个点的分布如下图所示.

我们通过观察数据的特征，仍然选用一样的神经网络，如下图所示.



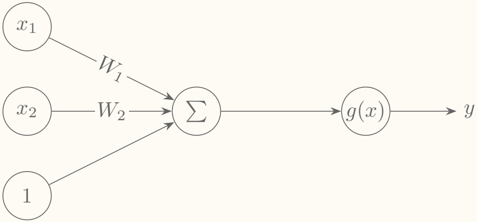
500 个点的分布图



TensorFlow 实战
分类问题

这次我们不用计算神经网络的表达式，直接照着神经网络写 TensorFlow 代码即可.

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

# 定义 x = (x1, x2), W = (W1, W2) 和 b = 1
x = tf.placeholder(tf.float32, [None, 2])
w = tf.Variable(tf.zeros([2, 1]))
b = tf.constant([1.])

# 定义神经网络输出 y = g(Wx + 1)
predict_type = tf.sigmoid((tf.matmul(x, w) + b) * 10)
# 定义真实输出
actual_type = tf.placeholder(tf.float32, [None, 1])
# 定义目标函数 ℓ(W) = −∑_{i=1}^{500} y_i ln g(Wx_i + 1) + (1 − y_i) ln (1 − g(Wx_i + 1))
loss = -tf.reduce_sum(actual_type * tf.log(predict_type) + (1 - actual_type) *
 ↪  tf.log(1 - predict_type))
```



TensorFlow 实战

分类问题


```python
def load_training_data(path): # 加载训练数据 x 和 y
  with open(path, 'r', encoding = 'utf8') as file:
    next(file)
    lines = [[float(item) for item in line.split()] for line in file]
    xs = np.array([line[:2] for line in lines])
    types = np.array([line[2] for line in lines])[:, None]
  return xs, types

with tf.Session() as session:
  session.run(tf.global_variables_initializer()) # 初始化变量
  train_step = tf.train.GradientDescentOptimizer(0.003).minimize(loss) # 梯度下降法
  xs, types = load_training_data('./training.dat') # 加载训练数据 x 和 y
  for i in range(1000): # 训练 1000 次
    session.run(train_step, feed_dict = {x:xs, actual_type:types}) # 填充训练数据
    # 打印目标函数值
    if i % 10 == 0: print(session.run(loss, feed_dict = {x:xs, actual_type:types}))
  print(session.run(w, feed_dict = {x:xs, actual_type:types})) # 打印最终结果
```

TensorFlow 实战

分类问题

最终, 参数 W_1 和 W_2 收敛于点 $(1.07, -3.10)$, 也是非常理想的结果.

TensorFlow 实战
分类问题

最终, 参数 W_1 和 W_2 收敛于点 $(1.07, -3.10)$, 也是非常理想的结果.

我们还可以换一个目标函数, 用模型预测错误的数量作为目标函数, 其表达式如下所示:

$$\ell(\boldsymbol{W}) = \sum_{x_1, x_2, y} \left| H(W_1 x_1 + W_2 x_2 + 1)) - y \right|,$$

TensorFlow 实战
分类问题

最终，参数 W_1 和 W_2 收敛于点 $(1.07, -3.10)$，也是非常理想的结果.

我们还可以换一个目标函数，用模型预测错误的数量作为目标函数，其表达式如下所示:

$$\ell(\boldsymbol{W}) = \sum_{x_1, x_2, y} \left| H(W_1 x_1 + W_2 x_2 + 1)) - y \right|,$$

由于绝对值不可微，

$$\text{上式} = \sum_{x_1, x_2, y} \left(H(W_1 x_1 + W_2 x_2 + 1)) - y \right)^2,$$

TensorFlow 实战
分类问题

最终，参数 W_1 和 W_2 收敛于点 $(1.07, -3.10)$，也是非常理想的结果.

我们还可以换一个目标函数，用模型预测错误的数量作为目标函数，其表达式如下所示:

$$\ell(\boldsymbol{W}) = \sum_{x_1,x_2,y} \left| H(W_1x_1 + W_2x_2 + 1)) - y \right|,$$

由于绝对值不可微，

$$上式 = \sum_{x_1,x_2,y} \left(H(W_1x_1 + W_2x_2 + 1)) - y \right)^2,$$

Sigmoid 函数替换，

$$\approx \sum_{x_1,x_2,y} \left(g(W_1x_1 + W_2x_2 + 1)) - y \right)^2.$$

TensorFlow 实战
分类问题

最终，参数 W_1 和 W_2 收敛于点 $(1.07, -3.10)$，也是非常理想的结果.

我们还可以换一个目标函数，用模型预测错误的数量作为目标函数，其表达式如下所示:

$$\ell(\boldsymbol{W}) = \sum_{x_1,x_2,y} \left| H(W_1x_1 + W_2x_2 + 1)) - y \right|,$$

由于绝对值不可微，

$$\text{上式} = \sum_{x_1,x_2,y} \left(H(W_1x_1 + W_2x_2 + 1)) - y \right)^2,$$

Sigmoid 函数替换，

$$\approx \sum_{x_1,x_2,y} \left(g(W_1x_1 + W_2x_2 + 1)) - y \right)^2.$$

我们修改目标函数的表达式，

```
loss = tf.reduce_sum(tf.square(actual_type - predict_type))
```

TensorFlow 实战

分类问题

最终, 参数 W_1 和 W_2 收敛于点 $(1.07, -3.10)$, 也是非常理想的结果.

我们还可以换一个目标函数, 用模型预测错误的数量作为目标函数, 其表达式如下所示:

$$\ell(\boldsymbol{W}) = \sum_{x_1,x_2,y} \left|H(W_1x_1 + W_2x_2 + 1)) - y\right|,$$

由于绝对值不可微,
$$\text{上式} = \sum_{x_1,x_2,y} \left(H(W_1x_1 + W_2x_2 + 1)) - y\right)^2,$$

Sigmoid 函数替换,
$$\approx \sum_{x_1,x_2,y} \left(g(W_1x_1 + W_2x_2 + 1)) - y\right)^2.$$

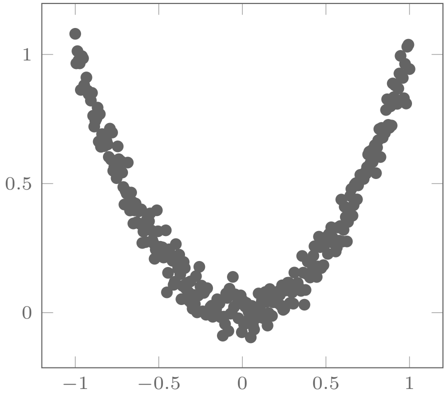
我们修改目标函数的表达式, 参数 W_1 和 W_2 收敛于点 $(1.07, -3.07)$, 效果拔群.

```
loss = tf.reduce_sum(tf.square(actual_type - predict_type))
```

TensorFlow 实战

拟合问题

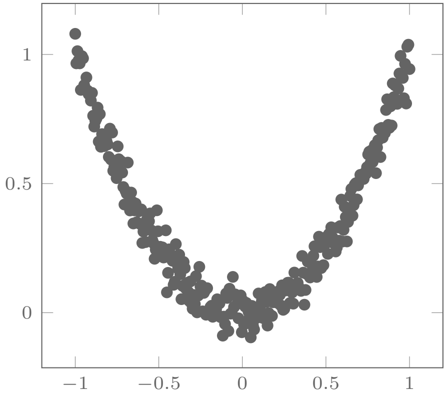
有 300 个点, 其分布如下图所示, 现在的目标是用一个神经网络, 来拟合这 300 个点.



TensorFlow 实战

拟合问题

有 300 个点, 其分布如下图所示, 现在的目标是用一个神经网络, 来拟合这 300 个点.



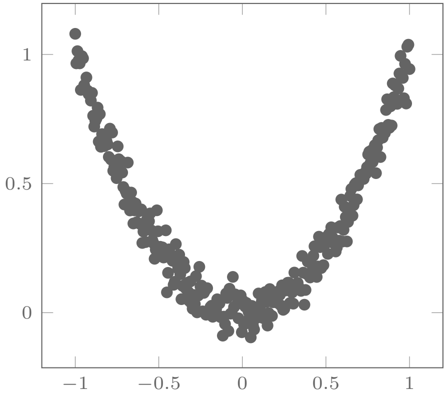
我们用一个两层网络来实现对这些离散点的拟合, 这两层网络的公式为:

$$
\begin{aligned}
\boldsymbol{x}_2 &= \mathrm{relu}(\boldsymbol{W}_1 x_1 + \boldsymbol{b}_1), \quad && \boldsymbol{W}_1 \in \mathbb{R}^{10\times 1},\ \boldsymbol{b}_1 \in \mathbb{R}^{10\times 1}, \\
\hat{y} &= \boldsymbol{W}_2 \boldsymbol{x}_2 + b_2, && \boldsymbol{W}_2 \in \mathbb{R}^{1\times 10},\ b_2 \in \mathbb{R}^{1\times 1}.
\end{aligned}
$$

TensorFlow 实战

拟合问题

有 300 个点，其分布如下图所示，现在的目标是用一个神经网络，来拟合这 300 个点.

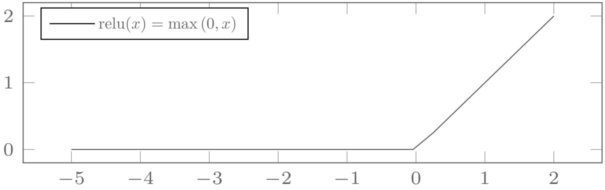


我们用一个两层网络来实现对这些离散点的拟合，这两层网络的公式为:

$$\boldsymbol{x}_2 = \mathrm{relu}(\boldsymbol{W}_1 x_1 + \boldsymbol{b}_1), \quad \boldsymbol{W}_1 \in \mathbb{R}^{10\times 1},\ \boldsymbol{b}_1 \in \mathbb{R}^{10\times 1},$$

$$\hat{y} = \boldsymbol{W}_2 \boldsymbol{x}_2 + b_2, \quad \boldsymbol{W}_2 \in \mathbb{R}^{1\times 10},\ b_2 \in \mathbb{R}^{1\times 1}.$$

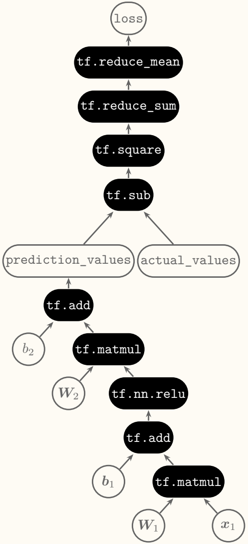
其中，函数 $\mathrm{relu}(\cdot)$ 被称为线性整流函数 (Rectified Linear Unit, ReLU)，图像如下所示.



TensorFlow 实战

拟合问题


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

x1 = tf.placeholder(tf.float32, [None, 1])
w1 = tf.Variable(tf.random_normal([1, 10]))
b1 = tf.Variable(tf.ones([1, 10]))
# $x_2 = \mathrm{relu}(W_1 x_1 + b_1),\ W_1 \in \mathbb{R}^{10\times 1},\ b_1 \in \mathbb{R}^{10\times 1}$
x2 = tf.nn.relu(tf.matmul(x1, w1) + b1)
w2 = tf.Variable(tf.random_normal([10, 1]))
b2 = tf.Variable(tf.ones([1, 1]))
# $\hat{y} = W_2 x_2 + b_2,\ W_2 \in \mathbb{R}^{1\times 10},\ b_2 \in \mathbb{R}^{1\times 1}$
predict_values = tf.matmul(x2, w2) + b2 # 预测值
actual_values = tf.placeholder(tf.float32, [None, 1]) # 真实值

# 目标函数 $\ell(W_1, b_1, W_2, b_2) = \frac{1}{n}\sum_{i=1}^{300}(y - \hat{y})^2$
loss = tf.reduce_mean(tf.reduce_sum(tf.square(actual_values - predict_values), reduction_indices = [1]))
```



TensorFlow 实战

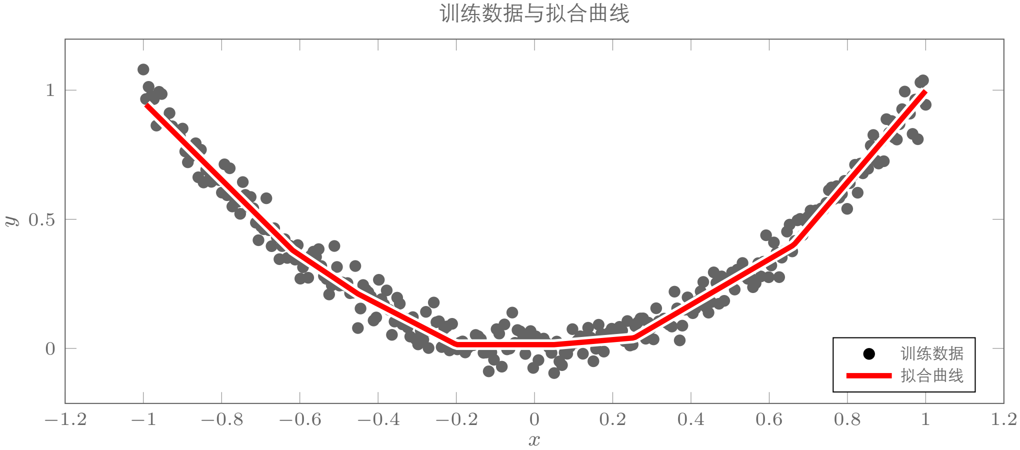
拟合问题


```python
def load_training_data(path): # 读取训练数据
    with open(path, 'r', encoding = 'utf8') as file:
        next(file)
        lines = [[float(item) for item in line.split(',')] for line in file]
    return (np.array([line[i] for line in lines])[:, None] for i in [0, 1])

with tf.Session() as session: # 初始化 Session
  session.run(tf.global_variables_initializer()) # 初始化变量

  (input_data, output_data) = load_training_data('./training_data.dat') # 加载训练数据
  train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss) # 设置目标函数

  for i in range(1000): # 训练 1000 次
    session.run(train_step, feed_dict = {x1:input_data, actual_values:output_data})
    if i % 50 == 0:
      print(session.run(loss, feed_dict = {x1:input_data, actual_values:output_data}))
```

TensorFlow 实战

拟合问题

训练 1000 次后，目标函数 $\ell(\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, b_2) = 2.48 \times 10^{-3}$，此时拟合函数的图像如图所示.



TensorFlow 实战

调试-TensorBoard

可以用 TensorBoard 来可视化 TensorFlow 的训练过程，帮助用户了解训练过程。启动 TensorBoard 首选需要记录训练过程数据：

```
# 建立数据流图 ...
tf.summary.histogram('w2', w2) # 需要记录 W2
tf.summary.scalar('loss', loss) # 需要记录目标函数 ℓ
merged = tf.summary.merge_all() # 合并需要记录的数据
with tf.Session() as session:
  writer = tf.summary.FileWriter('./tensorboard/') # 新建一个记录器
  writer.add_graph(session.graph) # 关联会话中的数据流图
  # 其他初始化 ...
  for i in range(1000):
    # 训练网络 ...
    writer.add_summary(session.run(merged, feed_dict = {x1:input_data,
     ↪   actual_values:output_data}), i)
    # 其他操作 ...
```

TensorFlow 实战

调试-TensorBoard

然后在终端中运行如下代码来启动 TensorBoard.

```
tensorboard --logdir=tensorboard
```

TensorFlow 实战

调试-TensorBoard

然后在终端中运行如下代码来启动 TensorBoard.

```
tensorboard --logdir=tensorboard
```

有时候系统会提示 `command not found: tensorboard`, 此时需要找到 TensorBorad 在哪. 首选在 Python 的终端中运行如下脚本, 获得 TensorBoard 的路径.

```
>>> import tensorboard
>>> print(tensorboard.__file__)
/path/of/your/tensorboard/__init__.py # ← 这个就是路径
```

TensorFlow 实战

调试-TensorBoard

然后在终端中运行如下代码来启动 TensorBoard.

```
tensorboard --logdir=tensorboard
```

有时候系统会提示 `command not found: tensorboard`, 此时需要找到 TensorBorad 在哪. 首选在 Python 的终端中运行如下脚本, 获得 TensorBoard 的路径.

```
>>> import tensorboard
>>> print(tensorboard.__file__)
/path/of/your/tensorboard/__init__.py # ← 这个就是路径
```

然后在系统终端运行如下脚本即可启动 TensorBoard. 查看当前训练的 TensorBoard, 可以用浏览器访问 `http://127.0.0.1:6006/` 这个地址.

```
python /path/of/your/tensorboard/main.py --logdir=tensorboard
```

TensorFlow 实战

调试-tfdbg

TensorFlow 中使用 Python 描述计算图，再使用 C++ 后端进行训练时就非常不容跟踪调试，TensorFlow 提供了 tfdbg 模块用于解决这个问题.

from: JieXiao's Blog — 《TensorFlow tfdbg》

TensorFlow 实战

调试-tfdbg

TensorFlow 中使用 Python 描述计算图，再使用 C++ 后端进行训练时就非常不容易跟踪调试，TensorFlow 提供了 tfdbg 模块用于解决这个问题.

from: JieXiao's Blog — 《TensorFlow tfdbg》

启动 tfdbg 很容易，只要在原始的代码中添加两行代码即可，再次运行便会进入 tfdbg 界面，在这里可以插入断点，单步运行 TensorFlow 的训练过程，查看每一个张量的值.

```
import tensorflow as tf
from tensorflow.python import debug as tf_debug # 第一行代码加在这
# 建立数据流图 ...
with tf.Session() as session:
  sess = tf_debug.LocalCLIDebugWrapperSession(sess) # 第二行代码加在这
  # 初始化 ...
  # 训练模型 ...
```

总结

- 以最直观的方式帮助大家理解神经网络，目标函数，梯度下降等概念；
- 介绍了机器学习的一般步骤：
  - 数据分析以及预处理，
  - 选择合适的模型，
  - 设定目标函数以及训练参数，
  - 验证以及调整模型；
- 介绍了 TensorFlow 中张量，节点，数据流图，会话等基本概念概念；
- 介绍了 TensorFlow 如何解决机器学习中分类和拟合这两个经典问题.

谢谢，欢迎提问

TensorFlow