



温州大學
WENZHOU UNIVERSITY

深度学习-神经网络的编程基础

黄海广 副教授

2023年03月

本章目录

2

- 01** 二分类与逻辑回归
- 02** 梯度下降
- 03** 计算图
- 04** 向量化

1.二分类与逻辑回归

3

01 二分类与逻辑回归

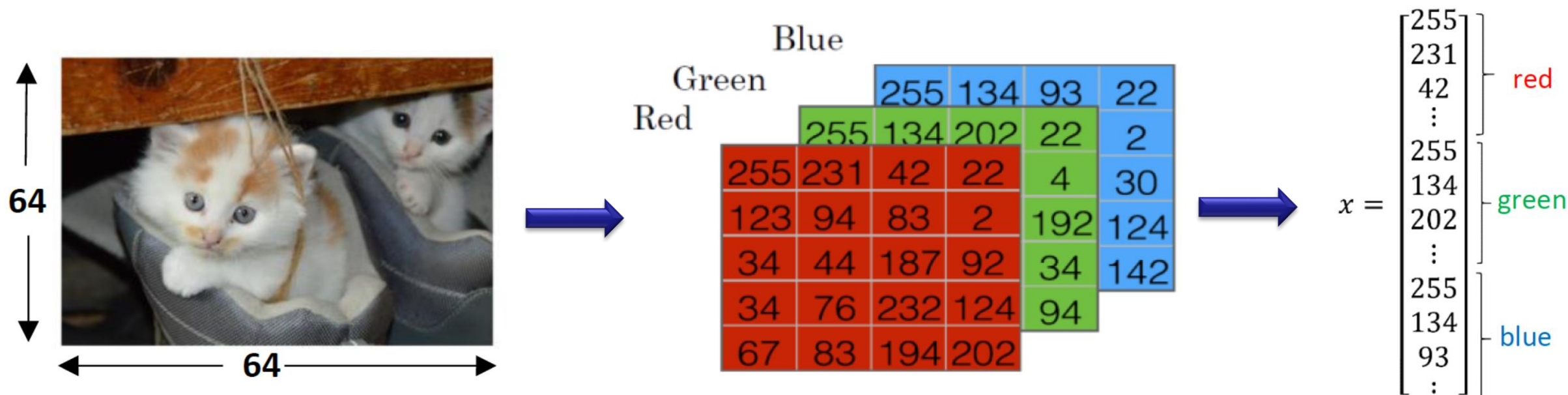
02 梯度下降

03 计算图

04 向量化

符号定义

4



x : 表示一个 n_x 维数据, 为输入数据, 维度为 $(n_x, 1)$;

y : 表示输出结果, 取值为 $(0,1)$;

$(x^{(i)}, y^{(i)})$: 表示第 i 组数据;

$X = [x^{(1)}, x^{(2)}, \dots, x^{(m)}]$: 表示所有的训练数据集的输入值, 放在一个 $n_x \times m$ 的矩阵中, 其中 m 表示样本数目;

$Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]$: 对应表示所有训练数据集的输出值, 维度为 $1 \times m$ 。

逻辑回归

5

Logistic Regression

经典的分类算法,简单、有效,
目前用到最多的机器学习分类算法之一。

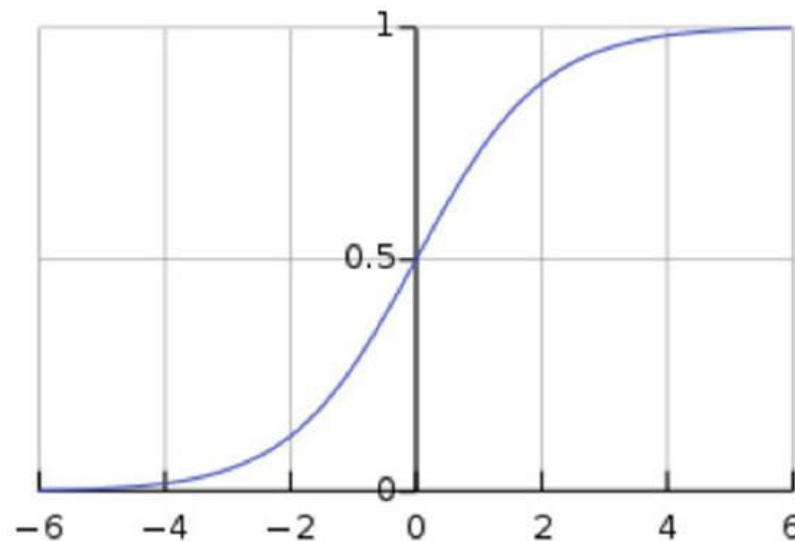
$\sigma(z)$ 代表一个常用的逻辑函数 (logistic function)
为S形函数 (Sigmoid function)

$$z = w^T x + b$$

$$\text{则: } \sigma(z) = \frac{1}{1 + e^{-z}}$$

合起来, 我们得到逻辑回归模型的假设函数:

$$L(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$



sigmoid 函数

当 $\sigma(z)$ 大于等于0.5时, 预测 $y=1$

当 $\sigma(z)$ 小于0.5时, 预测 $y=0$

逻辑回归

6

损失函数

$$L(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

\hat{y} 表示预测值

y 表示真实值

为了衡量算法在全部训练样本上的表现如何，我们需要定义一个算法的代价函数，算法的代价函数是对 m 个样本的损失函数求和然后除以 m ：

代价函数

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m \left(-y^{(i)} \log \hat{y}^{(i)} - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right)$$

逻辑回归的梯度下降

7

损失函数 $L(\hat{y}, y)$

设: $a = \hat{y}$

$$L(\hat{y}, y) = L(a, y) = -y\log(a) - (1 - y)\log(1 - a)$$

$$z = w^T x + b$$

因为 $\frac{dL(a, y)}{dz} = \frac{dL}{dz} = \left(\frac{dL}{da}\right) \cdot \left(\frac{da}{dz}\right)$, 并且 $\frac{da}{dz} = a \cdot (1 - a)$,

而 $\frac{dL(a, y)}{da} = \frac{dL}{da} = \left(-\frac{y}{a} + \frac{(1-y)}{(1-a)}\right)$, 因此将这两项相乘, 得到:

$$dz = \frac{dL(a, y)}{dz} = \frac{dL}{dz} = \left(\frac{dL}{da}\right) \cdot \left(\frac{da}{dz}\right) = \left(-\frac{y}{a} + \frac{(1-y)}{(1-a)}\right) \cdot a(1 - a) = a - y$$

逻辑回归的梯度下降

8

损失函数 $L(\hat{y}, y)$

设: $a = \hat{y}$

$$L(\hat{y}, y) = L(a, y) = -y\log(a) - (1 - y)\log(1 - a)$$

$$z = w^T x + b$$

因为 $\frac{dL(a, y)}{dz} = \frac{dL}{dz} = \left(\frac{dL}{da}\right) \cdot \left(\frac{da}{dz}\right)$, 并且 $\frac{da}{dz} = a \cdot (1 - a)$,

而 $\frac{dL(a, y)}{da} = \frac{dL}{da} = \left(-\frac{y}{a} + \frac{(1-y)}{(1-a)}\right)$, 因此将这两项相乘, 得到:

$$dz = \frac{dL(a, y)}{dz} = \frac{dL}{dz} = \left(\frac{dL}{da}\right) \cdot \left(\frac{da}{dz}\right) = \left(-\frac{y}{a} + \frac{(1-y)}{(1-a)}\right) \cdot a(1 - a) = a - y$$

2.梯度下降

9

01 二分类与逻辑回归

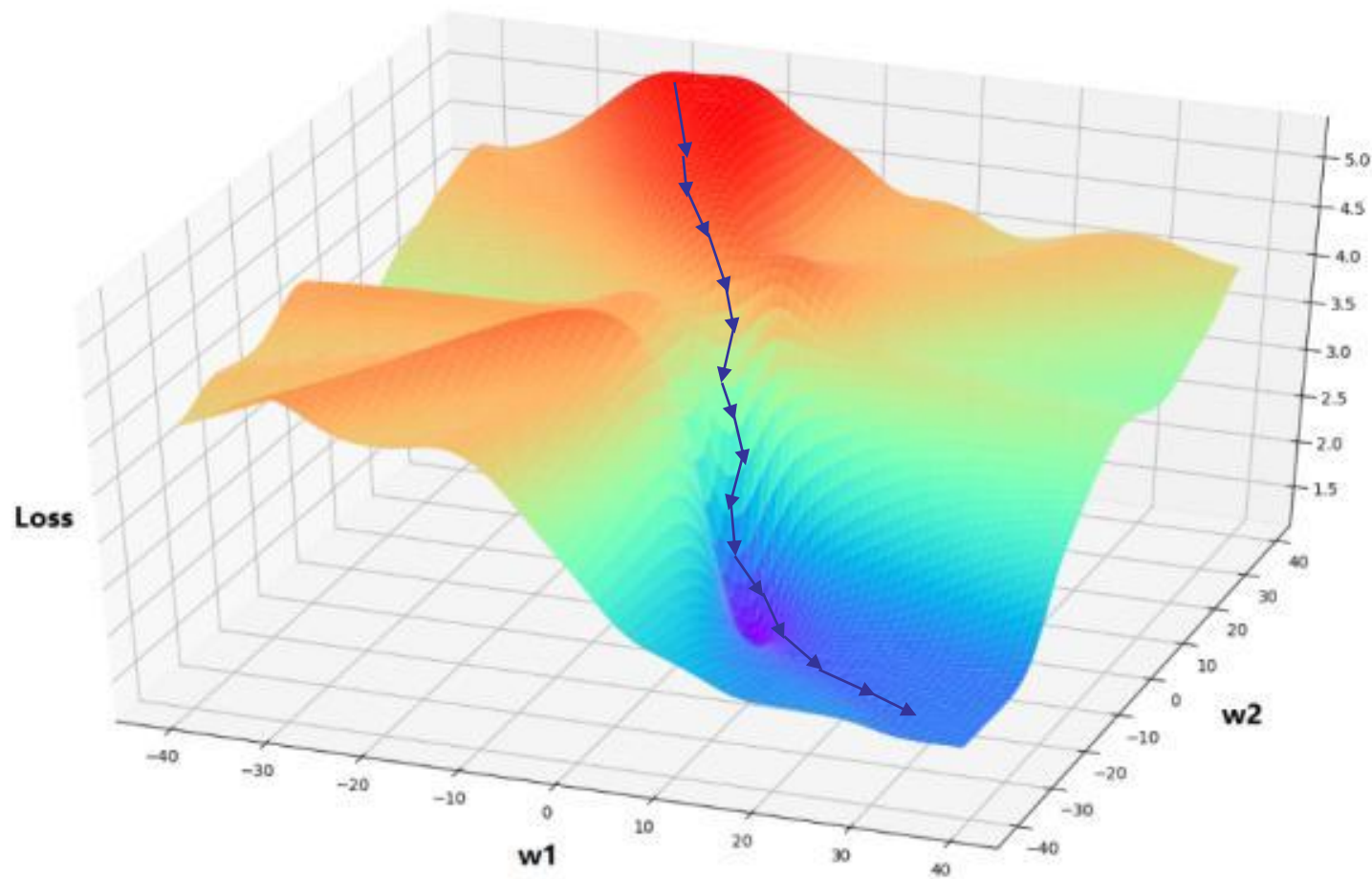
02 梯度下降

03 计算图

04 向量化

梯度下降

10



学习率

α

步长

梯度下降的三种形式

11

批量梯度下降 (Batch Gradient Descent, BGD)

梯度下降的每一步中，都用到了所有的训练样本

随机梯度下降 (Stochastic Gradient Descent, SGD)

梯度下降的每一步中，用到一个样本，在每一次计算之后便更新参数，而不需要首先将所有的训练集求和

小批量梯度下降 (Mini-Batch Gradient Descent, MBGD)

梯度下降的每一步中，用到了一定批量的训练样本

梯度下降的三种形式

12

批量梯度下降 (Batch Gradient Descent)

梯度下降的每一步中，都用到了所有的训练样本

参数更新

学习率

梯度

$$w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^m \left((h(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \right)$$

(同步更新 w_j , $(j=0,1,...,n)$)

梯度下降的三种形式

13

随机梯度下降 (Stochastic Gradient Descent)

推导 $w = w - \alpha \cdot \frac{\partial J(w)}{\partial w}$ $h(x) = w^T X = w_0 x_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$

$$J(w) = \frac{1}{2} (h(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial w_j} J(w) = \frac{\partial}{\partial w_j} \frac{1}{2} (h(x^{(i)}) - y^{(i)})^2 = 2 \cdot \frac{1}{2} (h(x^{(i)}) - y^{(i)}) \cdot \frac{\partial}{\partial w_j} (h(x^{(i)}) - y^{(i)})$$

$$= (h(x^{(i)}) - y^{(i)}) \cdot \frac{\partial}{\partial w_j} \left(\sum_{i=0}^n (w_i x_i^{(i)} - y^{(i)}) \right)$$

$$= (h(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

梯度下降的三种形式

14

随机梯度下降 (Stochastic Gradient Descent)

梯度下降的每一步中，用到一个样本，在每一次计算之后便更新参数，而不需要首先将所有的训练集求和

参数更新

$$w_j := w_j - \alpha (h(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(同步更新 w_j , $(j=0,1,\dots,n)$)

梯度下降的三种形式

15

小批量梯度下降 (Mini-Batch Gradient Descent)

梯度下降的每一步中，用到了一定批量的训练样本

每计算常数 b 次训练实例，便更新一次参数 w

参数更新

$$w_j := w_j - \alpha \frac{1}{b} \sum_{k=i}^{i+b-1} (h(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(同步更新 w_j , $(j=0,1,...,n)$)

$b=1$ (随机梯度下降,SGD)
 $b=m$ (批量梯度下降,BGD)
 $b=batch_size$, 通常是2的指数倍, 常见有32,64,128等。
(小批量梯度下降,MBGD)

逻辑回归的梯度下降

16

小批量梯度下降 (Mini-Batch Gradient Descent)

梯度下降的每一步中，用到了一定批量的训练样本

每计算常数 b 次训练实例，便更新一次参数 w

参数更新

$$w_j := w_j - \alpha \frac{1}{b} \sum_{k=i}^{i+b-1} (h(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(同步更新 w_j , $(j=0,1,...,n)$)

$b=1$ (随机梯度下降,SGD)
 $b=m$ (批量梯度下降,BGD)
 $b=batch_size$, 通常是2的指数倍, 常见有32,64,128等。
(小批量梯度下降,MBGD)

3.计算图

17

01 二分类与逻辑回归

02 梯度下降

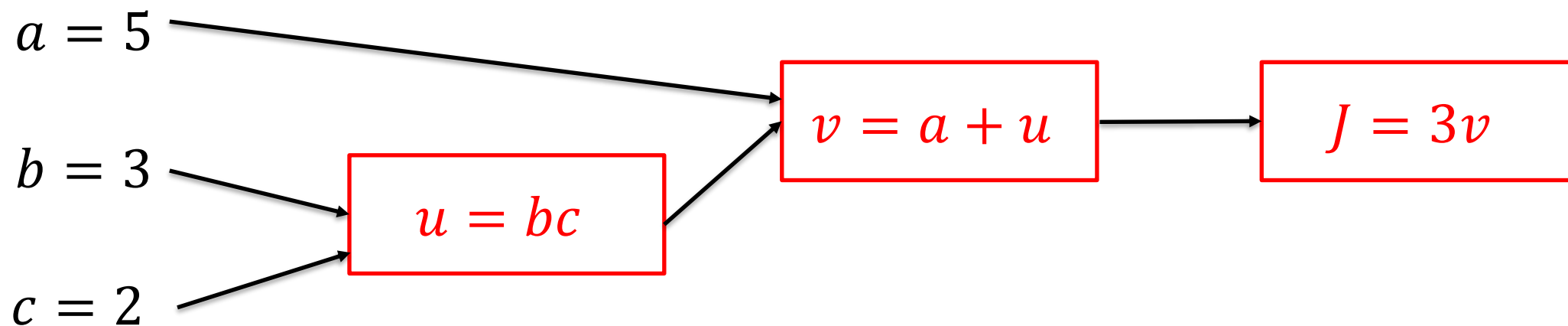
03 计算图

04 向量化

3.计算图

18

$$J(a, b, c) = 3(a + bc), a = 5, b = 3, c = 2$$



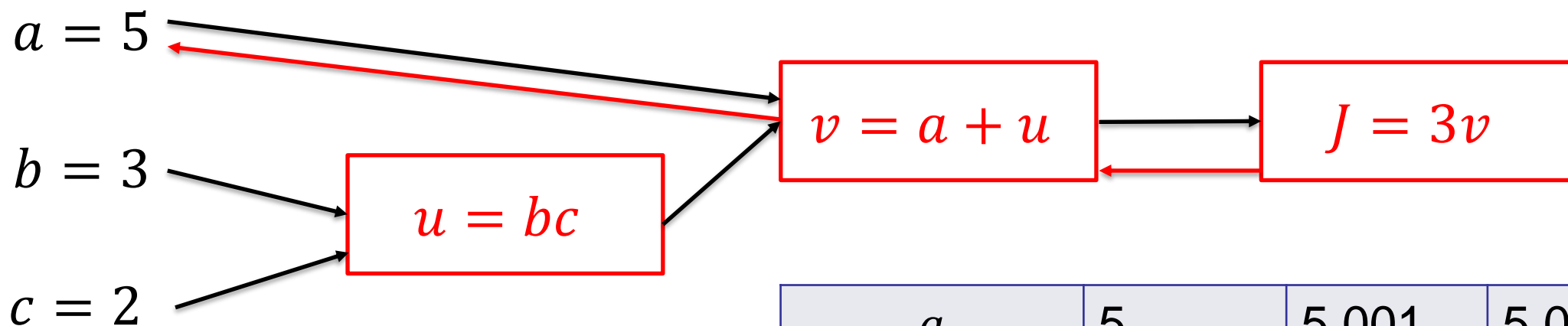
u	6
v	11
J	33

3.计算图

19

$$J(a, b, c) = 3(a + bc)$$

$$\frac{dJ}{du} = \frac{dJ}{dv} \frac{dv}{du}, \quad \frac{dJ}{db} = \frac{dJ}{du} \frac{du}{db}, \quad \frac{dJ}{da} = \frac{dJ}{du} \frac{du}{da}$$



$$\frac{dJ}{dv} = 3, \quad \frac{dv}{da} = 1$$

a	5	5.001	5.002
u	6	6.001	6.002
v	11	11.001	11.002
J	33	33.003	33.006

3.静态图与动态图

20

- 动态图：运算与搭建同时进行
- 静态图：先搭建图，后运算

根据计算图搭建方式，可将计算图分为动态图和静态图

PyTorch 是支持**动态图**的，可以在进行运算的同时进行

TensorFlow 是支持**静态图**的，需要在计算之前，先进行搭建（TensorFlow 2.X引入了动态图）

4.静态图与动态图

21

TensorFlow:先搭建所有的计算图之后，再把数据输入进去

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
w2 = tf.Variable(tf.random_normal((H, D)))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

learning_rate = 1e-5
new_w1 = w1.assign(w1 - learning_rate * grad_w1)
new_w2 = w2.assign(w2 - learning_rate * grad_w2)
updates = tf.group(new_w1, new_w2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    losses = []
    for t in range(50):
        loss_val, _ = sess.run([loss, updates],
                               feed_dict=values)
```

创建图

运行每次迭代

PyTorch:动态图的搭建是根据每一步的计算搭建的

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

每次迭代中创建图

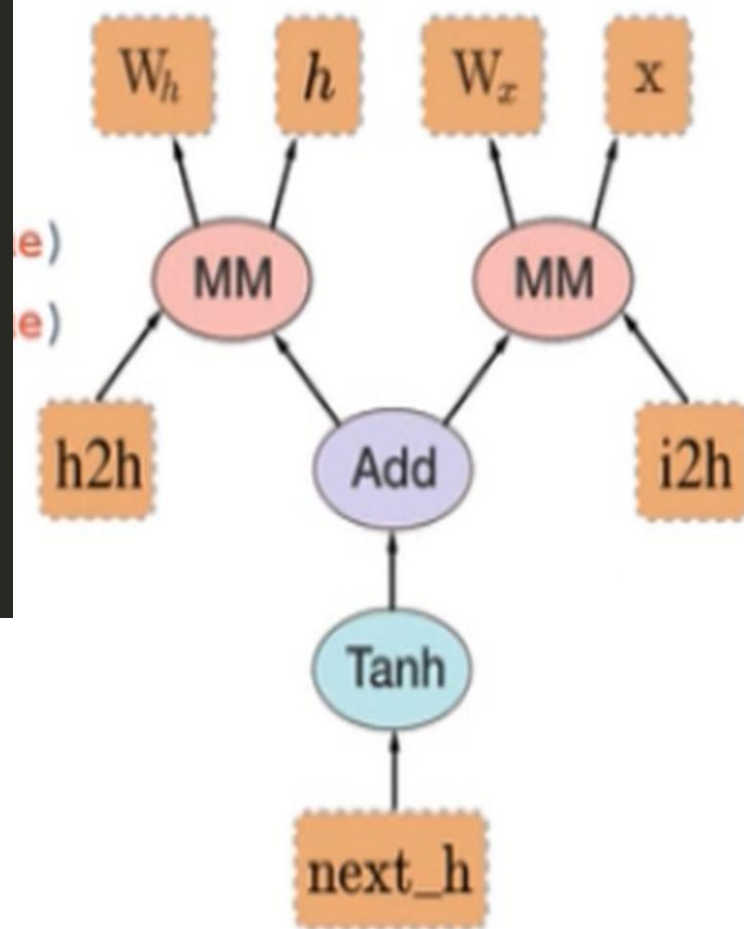
4.静态图与动态图

22

```
W_h = torch.randn(20, 20, requires_grad=True) #先创建四个张量
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)

h2h = torch.mm(W_h, prev_h.t()) #将W_h和prev_h进行相乘, 得到一个新张量h2h
i2h = torch.mm(W_x, x.t()) #将W_x和x进行相乘, 得到一个张量i2h
next_h = h2h + i2h #创建加法操作
next_h = next_h.tanh() #使用激活函数

loss = next_h.sum() #计算损失函数
loss.backward() #梯度反向传播
```



4.向量化

23

01 二分类与逻辑回归

02 梯度下降

03 计算图

04 向量化

4.向量化

24

```
import numpy as np #导入numpy库
a = np.array([1,2,3,4]) #创建一个数据a
print(a)
```

```
[1 2 3 4]
```

```
import time #导入时间库
a = np.random.rand(1000000)
b = np.random.rand(1000000) #通过round随机得到两个一百万维度的数组
tic = time.time() #现在测量一下当前时间
#向量化的版本
c = np.dot(a,b)
toc = time.time()
print('Vectorized version:' + str(1000*(toc-tic)) + 'ms') #打印一下向量化的版本的时间

#继续增加非向量化的版本
c = 0
tic = time.time()
for i in range(1000000):
    c += a[i]*b[i]
toc = time.time()
print(c)
print('For loop:' + str(1000*(toc-tic)) + 'ms') #打印for循环的版本的时间
```

```
Vectorized version:32.90534019470215ms
```

```
250144.2759995963
```

```
For loop:718.0795669555664ms
```

非向量化版本的for循环花费了大约718毫秒，向量化版本花费了大约33毫秒。

举例：如果你想计算向量 $u = Av$ ，
矩阵乘法的定义就是： $u_i = \sum_j A_{ij}v_j$ ，
用非向量化实现， $u = \text{np.zeros}(n, 1)$ ，
并且通过两层循环 $\text{for}(i): \text{for}(j):$ ，得到
 $u[i] = u[i] + A[i][j] * v[j]$ 。现在就有了
 i 和 j 的两层循环，这就是非向量化。
向量化方式就可以用 $u = \text{np.dot}(A, v)$ ，

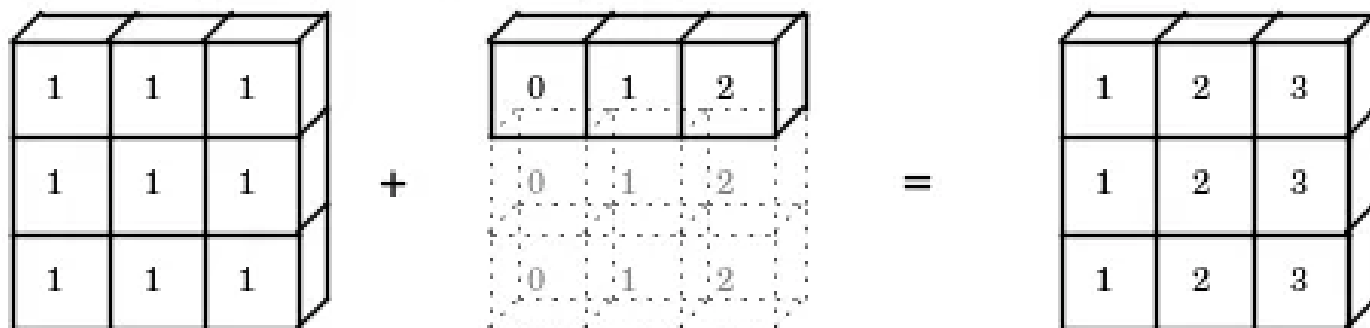
4.向量化-Python广播

25

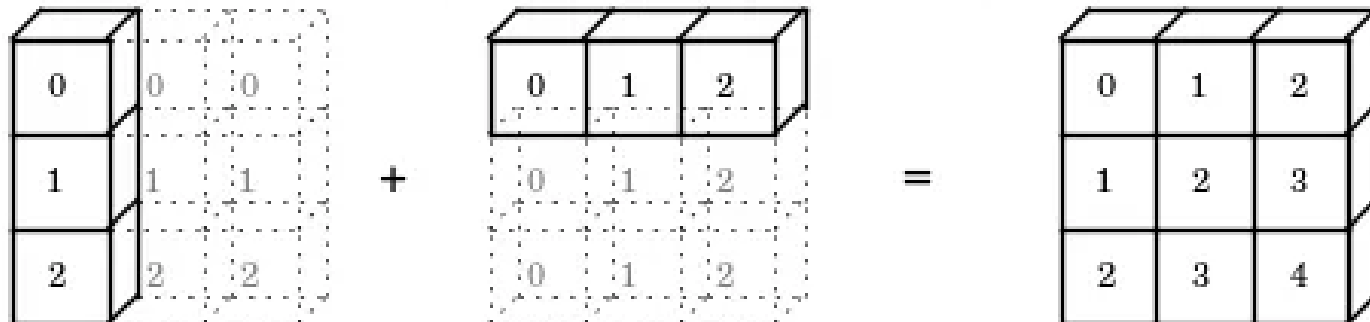
`np.arange(3) + 5`



`np.ones((3, 3)) + np.arange(3)`



`np.arange(3).reshape((3, 1)) + np.arange(3)`



1. IAN GOODFELLOW等, 《深度学习》, 人民邮电出版社, 2017
2. Andrew Ng, <http://www.deeplearning.ai>

谢谢!

