Contents

contents

1 1. logistic regression as a neural network

1.1 1.1. binary classification

维度为 (64, 64, 3) 的图片 ===> img vector: x= 维度为 (64*64*3=12288, 1) 的列向量。($n_x = 12288$)

$$(x, y), x \in R^{n_x}, y \in \{0, 1\}$$

$m = m_{train}$ 个训练样本: $(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)}), m_{test}$ 个测试样本。

X 表示一个 $n_x * m$ 的训练样本矩阵, 在 python 里就是 `X.shape=(n_x,m)` Y 表示一个 $1 * m$ 的向量, 在 python 里是 `Y.shape=(1,m)`

1.2 1.2. logistic regression

given x, want $\hat{y} = P(y = 1|x), x \in R^{n_x}$

params: $w \in R^{n_x}, b \in R$

output: $\hat{y} = \sigma(w^T x + b), \sigma(z) = \frac{1}{1+e^{(-z)}}$

1.3 1.3. logistic regression cost function

$(x^{(i)}, y^{(i)})$ 表示第 i 个样本。

Loss(error) function 只针对一条训练样本:

- square error 的 loss function:

$$L(\hat{y}, y) = 1/2 * (\hat{y} - y)^2$$

- logistic regression 的 loss function:

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

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

if $y = 0, L(\hat{y}, y) = -log(1 - \hat{y})$, want $\hat{y}$ small

Cost function 针对全体训练样本:

$$J(w, b) = 1/m \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -1/m \sum_{i=1}^{m} [y^{(i)} log\hat{y}^{(i)} + (1 - y^{(i)}) log(1 - \hat{y}^{(i)})]$$

1.4 1.4. gradient descent

lr 的 $J(w, b)$ 是一个凸函数, 所以有全局最优。因为有全局最优, 所以 lr 的初始化一般是 0, 不用随机。梯度下降: 不断重复 $w = w - \alpha \frac{dJ(w)}{dw}$ 直到收敛。后续, 用 dw 来指代 $\frac{dJ(w)}{dw}$。梯度下降的公式:

$$w = w - \alpha dw$$

$$b = b - \alpha db$$

1.5 1.5. derivatives

derivative = slope, 就是 $dy/dx = \Delta y/\Delta x$

1.6 1.6. more derivative examples

$f(a) = log_e(a) = ln(a), df(a)/da = \frac{1}{a}$

1.7 1.7. computation graph

正向计算图算出输出，算每个参数的梯度就反向算。

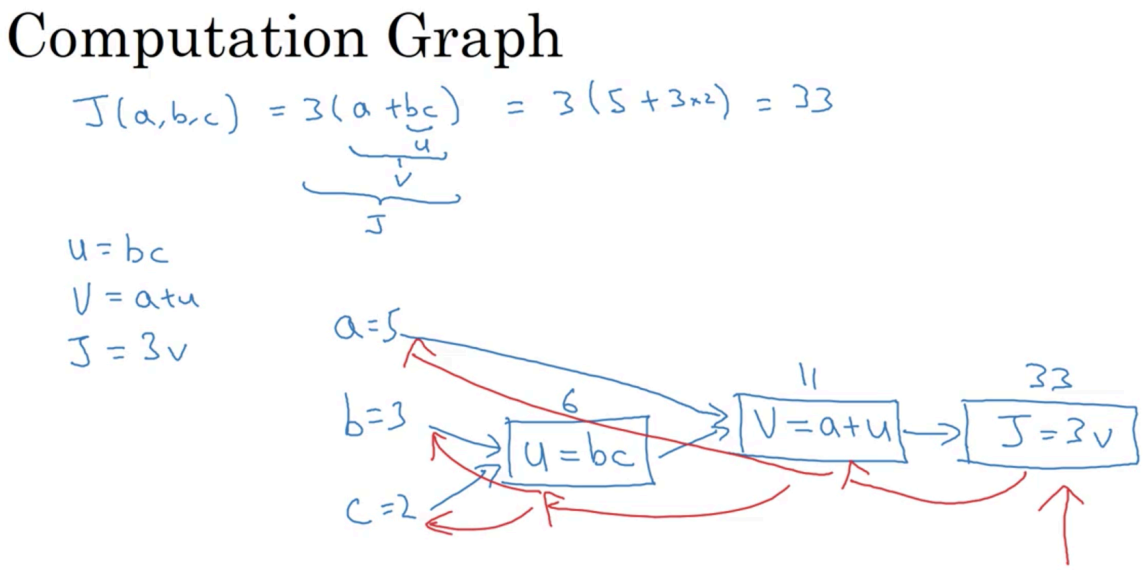


Figure 1: computation graph

1.8 1.8. derivatives with a computation graph

$$J = 3v, v = a + u, u = bc$$

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

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

$$if\ b = 2, then\ \frac{dJ}{dc} = \frac{dJ}{dv}\frac{dv}{du}\frac{du}{dc} = 3 * 1 * b = 3 * 1 * 2 = 6$$

写代码时，将 $\frac{dFinalOutputVar}{dvar}$ 记为 $dvar$**(最后输出对这个变量的偏导)**

1.9 1.9. logistic regression gradient descent

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

$$\hat{y} = a = \sigma(z)$$

$$L(a, y) = -(ylog(a) + (1 - y)log(1 - a))$$

1.10 1.10. gradient descent on m examples

首先，根据 J 的公式，可以知道 dJ/dw1 其实就是对每个样本的 dw1 求和，然后/m。

每一次迭代，遍历 m 个样本，算出 J/dw1/dw2/db，然后用这些梯度去更新一次 w1/w2/b。

但这的 for loop 太多了。。所以我们需要 vectorization!

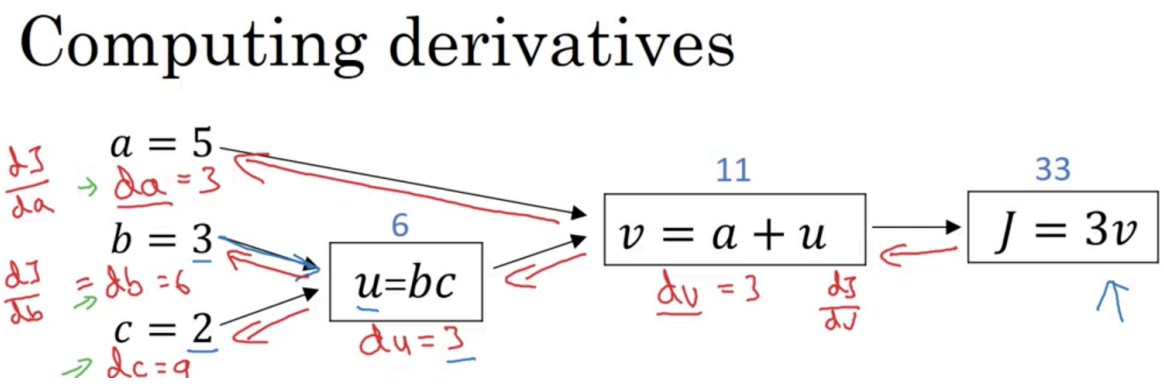


Figure 2: derivatives in computation graph

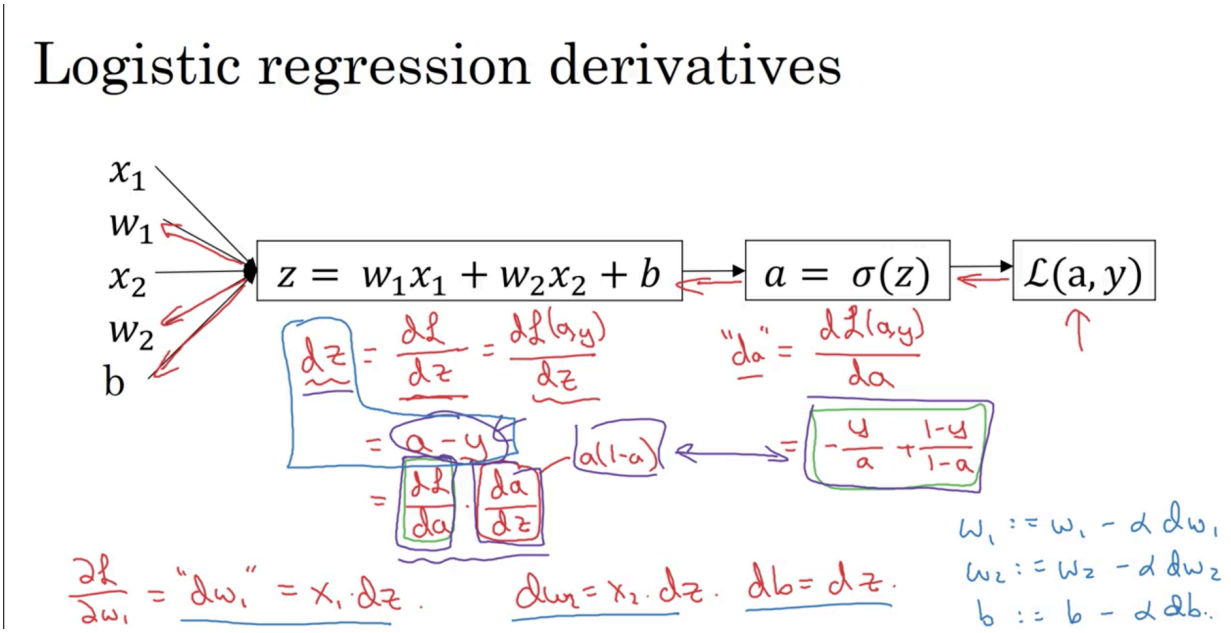


Figure 3: derivatives in computation graph in lr

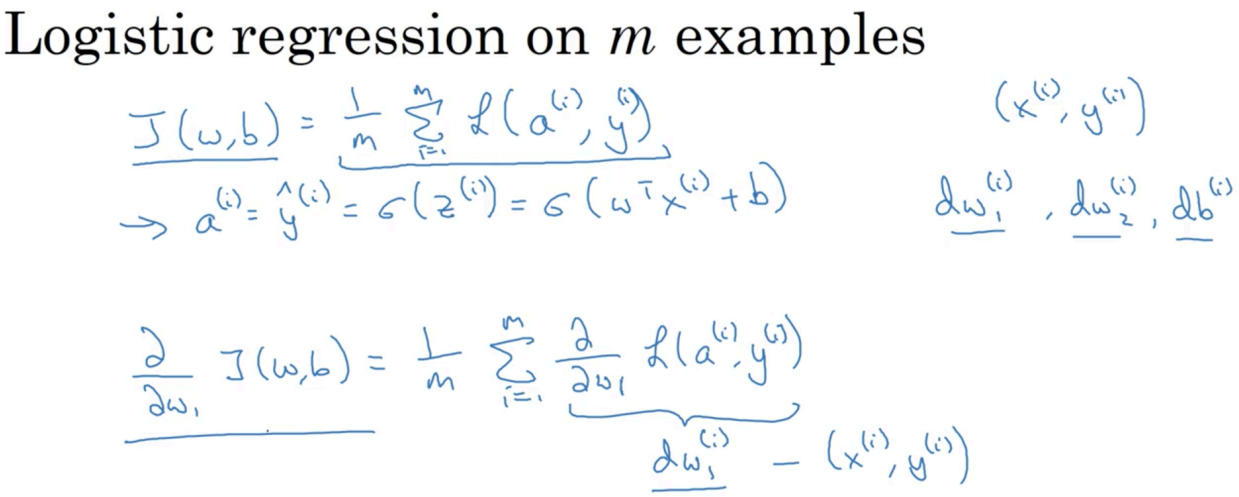


Figure 4: gradient_descent_lr_m_examples_djdw

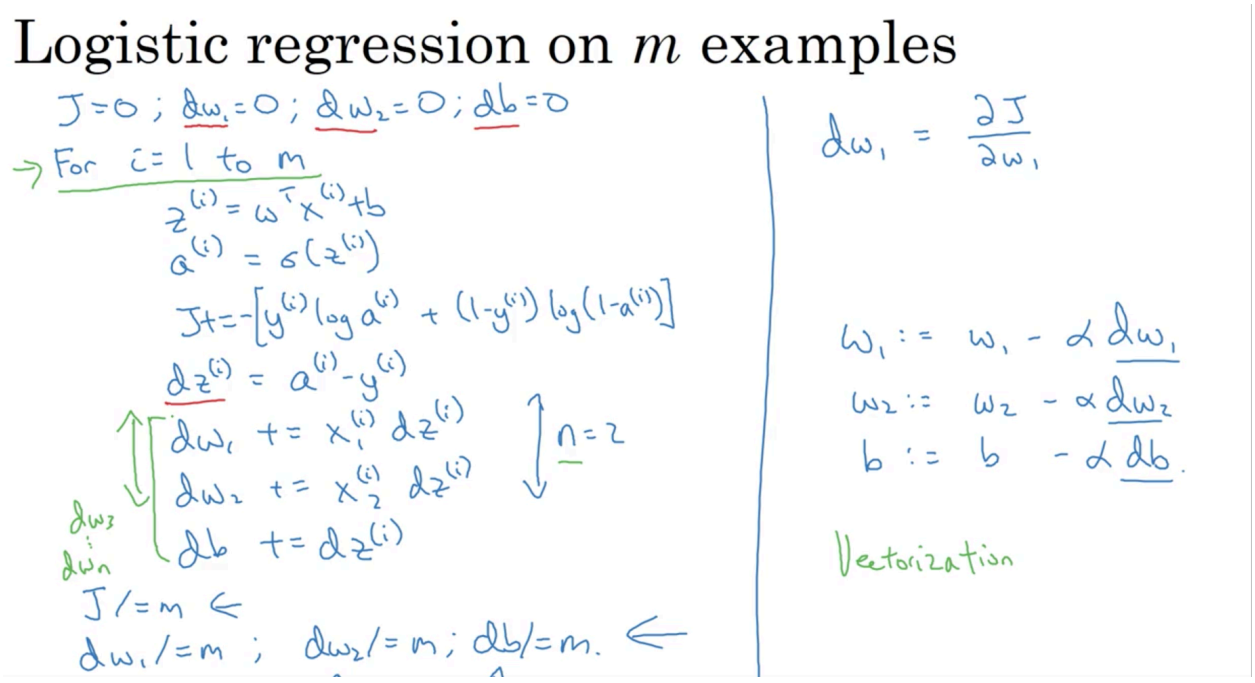


Figure 5: gradient_descent_lr_m_examples

2 2. python & vectorization

2.1 2.1. vectorization

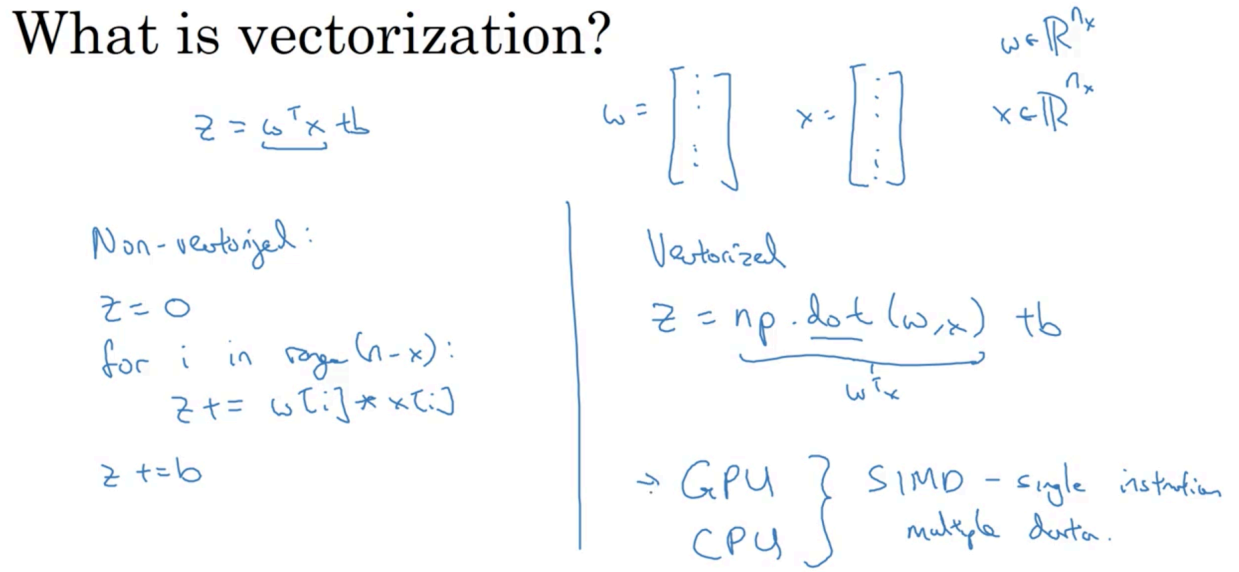
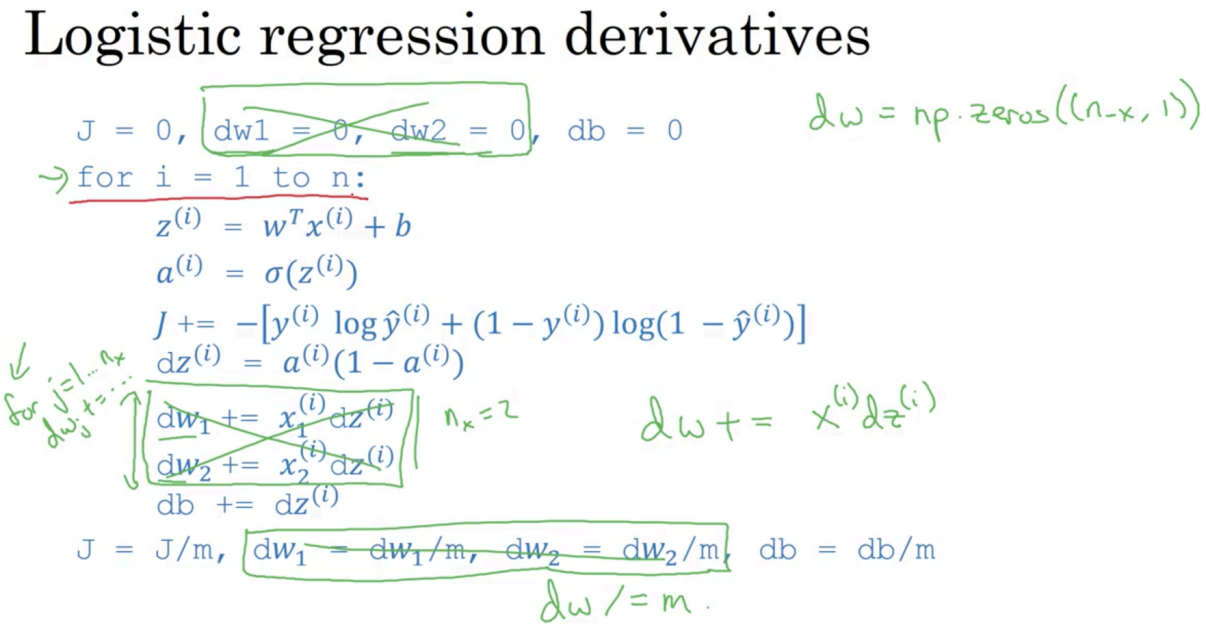


Figure 6: gradient_descent_lr_m_examples

对于两个 100w 维的向量进行点乘，vectorization(1.5ms) 比 for loop(470ms+) 快

2.2 2.2. more examples of vectorization



如上图，将 n_x 维的 dw 变为一个 np.array 即可干掉内层的 for loop。

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

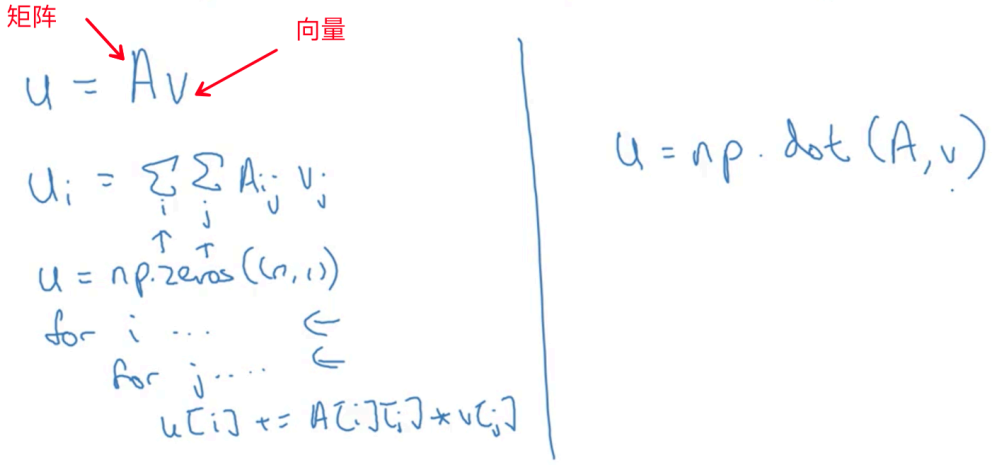


Figure 7: gradient_descent_lr_m_examples

Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector.

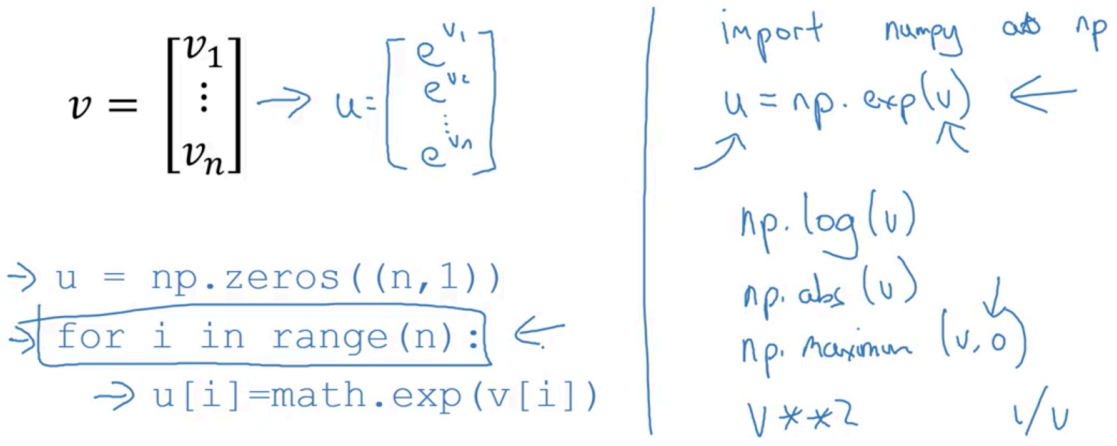


Figure 8: gradient_descent_lr_m_examples

2.3 2.3. vectorizing logistic regression

可见，整个求 Z 的过程可以变成一句话，而求 A 时，需要封装一个基于 numpy 的 sigmoid 函数。

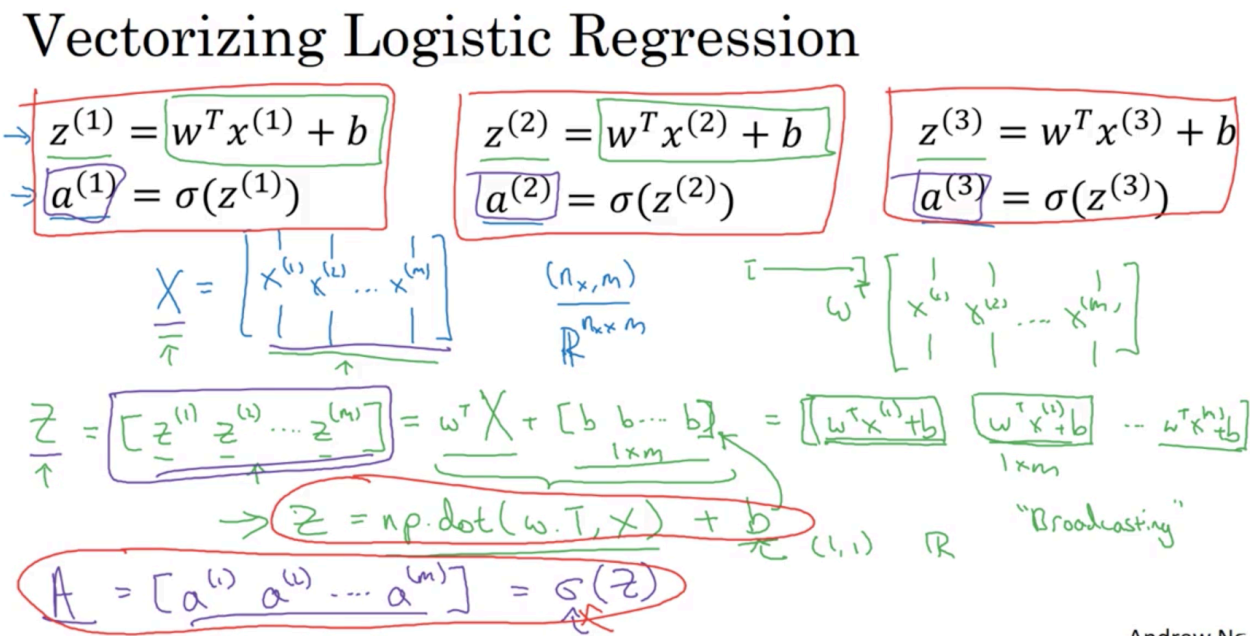


Figure 9: gradient_descent_lr_m_examples

2.4 2.4. vectorizing logistic regression's gradient output

2.5 2.5. broadcasting in python

```
A = ndarray([[1,2,3,4],[2,3,4,5],[3,4,5,6]]) # 3*4
calc = A.sum(axis=0) # A 的每列求和，得到 1*4
calc2 = A.sum(axis=1) # A 的每行求和，得到 3*1
A/calc.reshape(1,4) # 得到一个 3*4 的矩阵，就是 broadcasting。其实等价于 A/calc，但为了保险，可以调用 reshape(1,4) 来确保
```

小结：

2.6 2.6. a note on python/numpy vectors

numpy 的 broadcasting 文档

其实就两个准则：

- they are equal, or
- one of them is 1

```
a=np.random.randn(5) # a.shape=(5,) 是一个 vector(rank 1 array)，不是一个矩阵，所以 a.T 还是 (5,)，np.dot(a,a.T)=
```

```
b=np.random.randn(5,1) # a.shape=(5,1),  a.T.shape=(1,5), np.dot(a,a.T) 是一个 5*5 的，np.dot(a.T,a) 是一个
```

```
## 可以加一句：
```

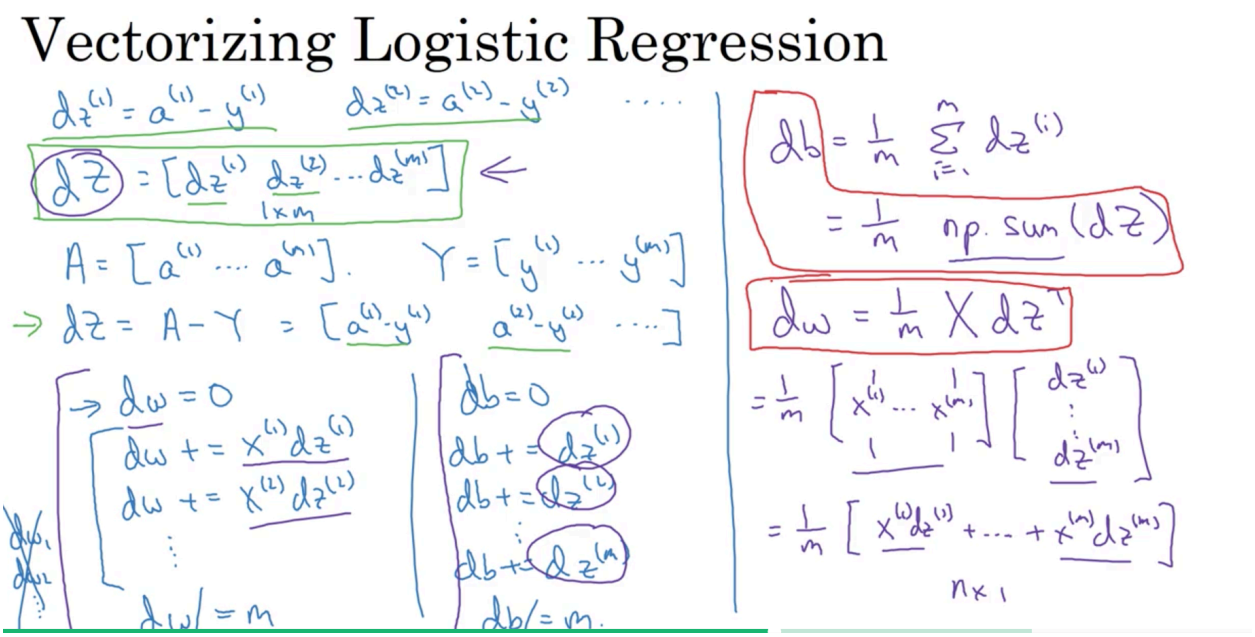


Figure 10: gradient_descent_lr_m_examples

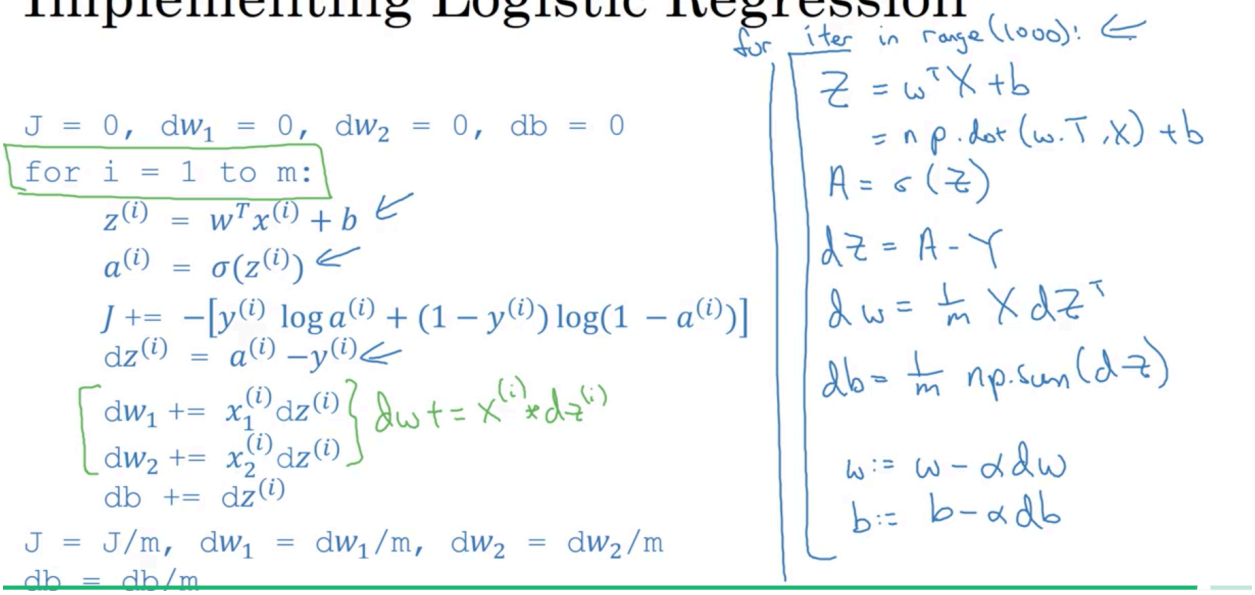


Figure 11: gradient_descent_lr_m_examples

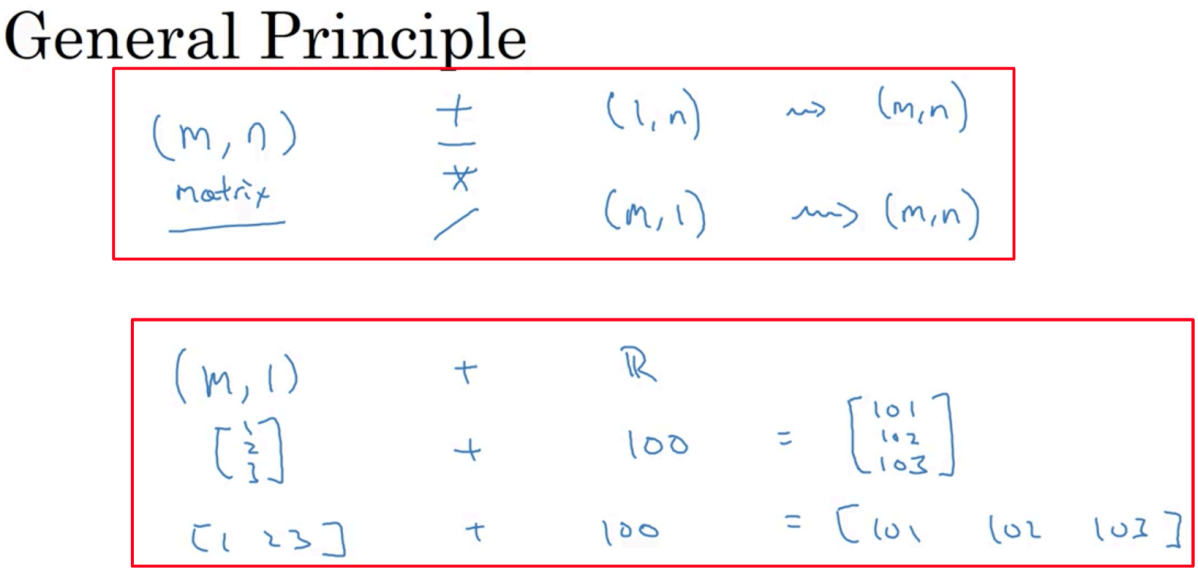


Figure 12: broadcasting

```
assert(a.shape == (5, 1))
## 如果不小心搞了个 rank 1 array, 也可以手动 a.reshape((5,1))=a.reshape(5,1)
```

2.7 2.7. quick tour of jupyter/ipython notebooks

2.8 2.8. explanation of logistic regression cost function

单个样本的 loss function, log 越大，loss 越小:

如果是 iid（独立同分布），那么，m 个样本的 cost function，其实就叫对数似然。对他求极大似然估计，其实就是对 m 个样本求每个 cost function 的 min:

2.9 2.9. programming assignments

squeeze

```
np.squeeze(a, axis=None)
## 删掉维数是一的部分，axis 可以是 Int/int 数组，表示只去掉指定下标的部分，如果该部分维数不是 1，会报错
x = np.array([[[0], [1], [2]]])
x.shape=(1,3,1)
np.squeeze(x)=array([0,1,2]) # shape=(3,)
np.squeeze(x, axis=(2,))=array([[0, 1, 2]]) # shape=(1,3)
```

把一个 shape 是 (a,b,c,d) 的 array 转成一个 type 是 (bcd,a) 的 array:

```
X_flatten = X.reshape(X.shape[0], -1).T
```

图片的预处理: + Figure out the dimensions and shapes of the problem (m_train, m_test, num_px, …) + Reshape the datasets such that each example is now a vector of size (num_px * num_px * 3, 1) + "Standardize" the data: 对图片而言，所有元素除以 255 就可以了

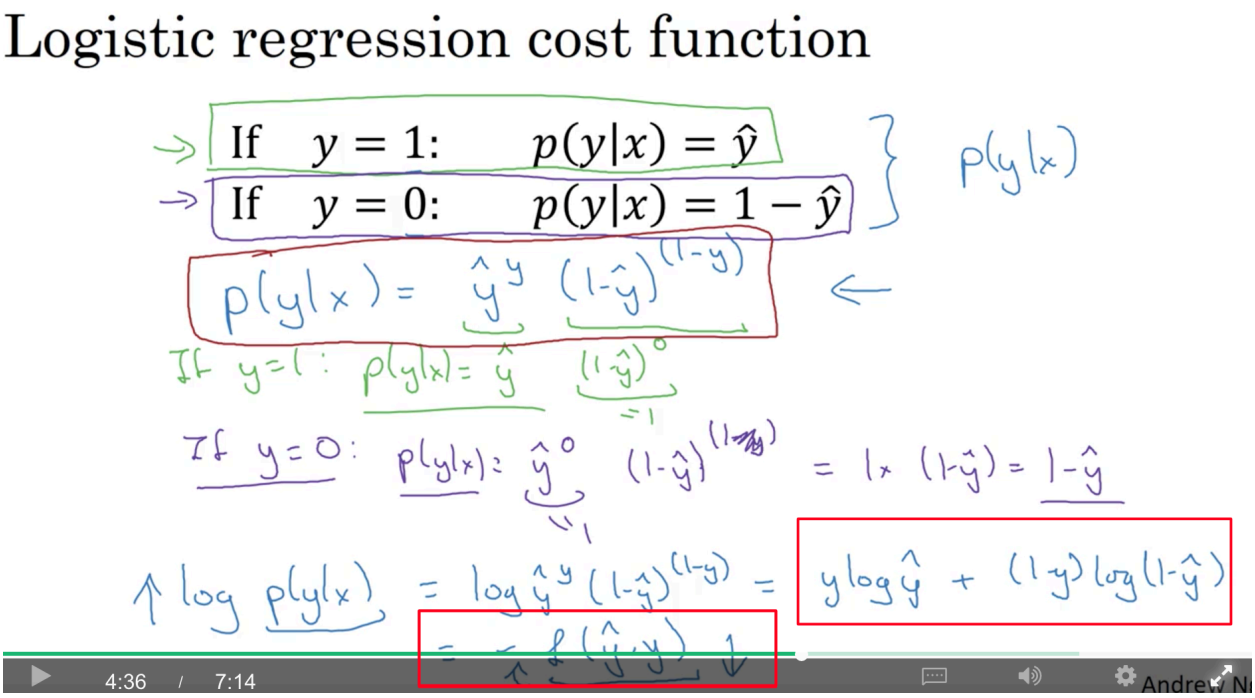


Figure 13: lr-loss

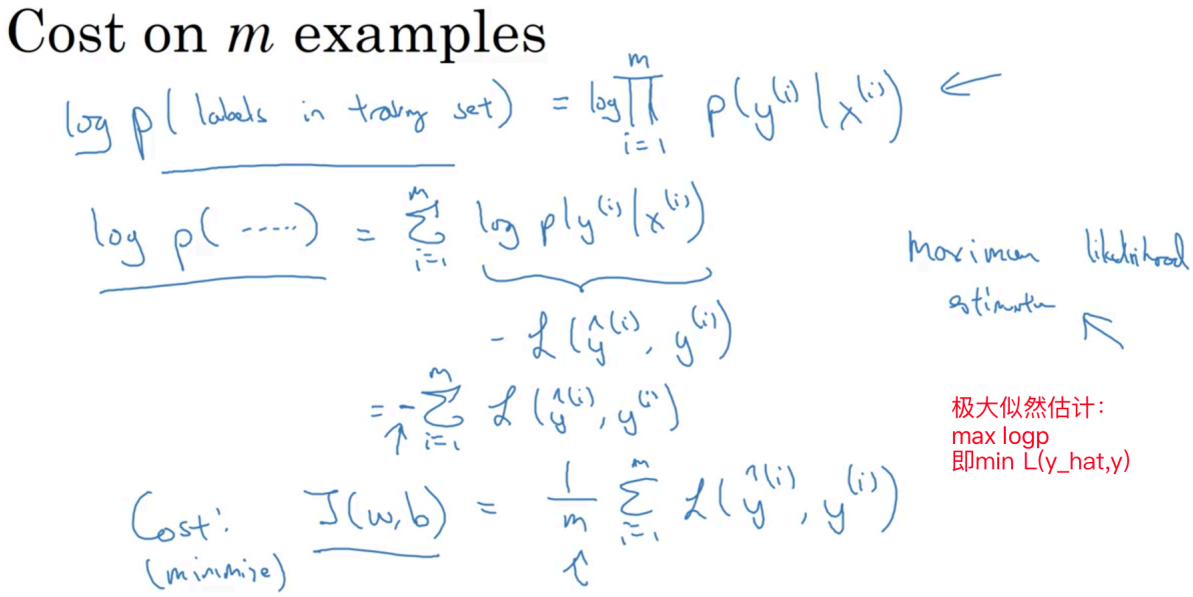


Figure 14: lr-cost-m-examples