

Join GitHub today

[Dismiss](#)

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

Branch: master ▾

[Find file](#)[Copy path](#)

[deep-learning-specialization-coursera](#) / [01-Neural-Networks-and-Deep-Learning](#) / [week2](#) / [02-vectorization.ipynb](#)



andersy005 add package version information

a5888a3 on Oct 14, 2018

[1 contributor](#)

[Raw](#)[Blame](#)[History](#)

426 lines (426 sloc) 9.44 KB

Table of Contents

- 1 Vectorization
- 2 Vectorizing Logistic Regression
- 3 Vectorizing Logistic Regression's Gradient Output
- 4 A note on python/numpy vectors

Vectorization

What is vectorization?

$$z = \omega^T x + b$$

Non-vectorized:

```

z = 0
for i in range(n-x):
    z += w[i] * x[i]
z += b

```

Vectorized:

$$z = \underbrace{\text{np.dot}(\omega, x)}_{\omega^T x} + b$$

$\omega \in \mathbb{R}^{n_x}$
 $x \in \mathbb{R}^{n_x}$

$\rightarrow \text{GPU}$ } SIMD - single instruction
 $\rightarrow \text{CPU}$ } multiple data.

```
In [1]: import numpy as np
```

```
In [2]: a = np.random.rand(1000000)
        b = np.random.rand(1000000)
```

```
In [3]: %time c = np.dot(a, b)
```

CPU times: user 1.62 ms, sys: 727 μ s, total: 2.34 ms
Wall time: 1.06 ms

```
In [4]: def loop():
        c = 0
        for i in range(1000000):
            c += a[i] * b[i]
```

```
In [5]: %time loop()
```

CPU times: user 316 ms, sys: 2.56 ms, total: 319 ms
Wall time: 317 ms

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

$$u = Av$$

$$u_i = \sum_j A_{ij} v_j$$

$$u = \text{np.zeros}(n, 1)$$

for i ...

for j ...

$$u[i] += A[i,j] * v[j]$$

$$u = \text{np.dot}(A, v)$$

Vectors and matrix valued functions

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

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

$\rightarrow u = \text{np.zeros}(n, 1)$

\rightarrow for i in range(n): \leftarrow
 $\rightarrow u[i] = \text{math.exp}(v[i])$

import numpy as np
 $u = \text{np.exp}(v)$ \leftarrow

$\text{np.log}(v)$

$\text{np.abs}(v)$

$\text{np.maximum}(v, 0)$

$v ** 2$ v / v

Logistic regression derivatives

$J = 0, \text{dw1} = 0, \text{dw2} = 0, db = 0$

$\text{dw} = \text{np.zeros}(n-x, 1)$

\rightarrow for i = 1 to n:

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J += -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

$$dz^{(i)} = a^{(i)}(1 - a^{(i)})$$

\leftarrow
 for j=1..n
 $\text{dw}_j += \dots$

$$\text{dw}_1 += x_1^{(i)} dz^{(i)}$$

$$\text{dw}_2 += x_2^{(i)} dz^{(i)}$$

$$db += dz^{(i)}$$

$$dw += x^{(i)} dz^{(i)}$$

$J = J/m, \text{dw1} = \text{dw1}/m, \text{dw2} = \text{dw2}/m, db = db/m$

$$dw /= m.$$

Vectorizing Logistic Regression

Vectorizing Logistic Regression

$$\begin{aligned}
 &\Rightarrow \boxed{z^{(1)} = w^T x^{(1)} + b} \quad \boxed{z^{(2)} = w^T x^{(2)} + b} \quad \boxed{z^{(3)} = w^T x^{(3)} + b} \\
 &\Rightarrow \boxed{a^{(1)} = \sigma(z^{(1)})} \quad \boxed{a^{(2)} = \sigma(z^{(2)})} \quad \boxed{a^{(3)} = \sigma(z^{(3)})}
 \end{aligned}$$

$X = \begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ 1 & 1 & \dots & 1 \end{bmatrix}$ $\frac{(n, m)}{\mathbb{R}^{n \times m}}$ $w^T = \begin{bmatrix} w^{(1)} & w^{(2)} & \dots & w^{(m)} \\ 1 & 1 & \dots & 1 \end{bmatrix}$

$$\vec{z} = \begin{bmatrix} z^{(1)} & z^{(2)} & \dots & z^{(m)} \end{bmatrix} = w^T X + \begin{bmatrix} b & b & \dots & b \end{bmatrix} = \begin{bmatrix} w^T x^{(1)} + b & w^T x^{(2)} + b & \dots & w^T x^{(m)} + b \end{bmatrix}$$

$\rightarrow z = \text{np.dot}(w.T, X) + b$ $\mathbb{R}^{(1,1)}$ "Broadcasting"

$$A = \begin{bmatrix} a^{(1)} & a^{(2)} & \dots & a^{(m)} \end{bmatrix} = \sigma(z)$$

Andrew Ng

Vectorizing Logistic Regression's Gradient Output

Implementing Logistic Regression

$J = 0, dw_1 = 0, dw_2 = 0, db = 0$

for $i = 1$ to m :

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J += -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$\begin{cases} dw_1 += x_1^{(i)} dz^{(i)} \\ dw_2 += x_2^{(i)} dz^{(i)} \end{cases} \quad dw += x^{(i)} * dz^{(i)}$$

$$db += dz^{(i)}$$

$$J = J/m, dw_1 = dw_1/m, dw_2 = dw_2/m$$

$$db = db/m$$

for iter in range(1000):

$$\begin{aligned}
 z &= w^T X + b \\
 &= \text{np.dot}(w.T, X) + b \\
 A &= \sigma(z) \\
 dz &= A - Y \\
 dw &= \frac{1}{m} X dz^T \\
 db &= \frac{1}{m} \text{np.sum}(dz) \\
 w &:= w - \alpha dw \\
 b &:= b - \alpha db
 \end{aligned}$$

Andrew Ng

A note on python/numpy vectors

```
In [6]: a = np.random.randn(5)
a
```

```
Out[6]: array([-0.91796822, -0.53903443,  1.00289266,  0.22272871, -
 0.35617949])
```

```
In [7]: a.shape
```

```
Out[7]: (5,)
```

```
In [8]: a.T
```

```
Out[8]: array([-0.91796822, -0.53903443,  1.00289266,  0.22272871, -
 0.35617949])
```

```
In [9]: np.dot(a, a.T)
```

```
Out[9]: 2.3154893533786054
```

```
In [10]: a = np.random.randn(5, 1)
a
```

```
Out[10]: array([[ -1.26834861],
 [ -0.254855  ],
 [ -1.37786229],
 [  0.18718574],
 [ -1.31341244]])
```

```
In [11]: a.shape
```

```
Out[11]: (5, 1)
```

```
In [12]: a.T
```

```
Out[12]: array([[ -1.26834861, -0.254855  , -1.37786229,  0.18718574,
 -1.31341244]])
```

```
In [13]: np.dot(a, a.T)
```

```
Out[13]: array([[ 1.60870819,  0.32324498,  1.74760972, -0.23741677,
 1.66586484],
 [ 0.32324498,  0.06495107,  0.35115509, -0.04770522,
 0.33472972],
 [ 1.74760972,  0.35115509,  1.8985045 , -0.25791618,
 1.80970147],
 [-0.23741677, -0.04770522, -0.25791618,  0.0350385 ,
 -0.24585208],
 [ 1.66586484,  0.33472972,  1.80970147, -0.24585208,
 1.72505223]])
```

Python/numpy vectors

`a = np.random.randn(5)`
`a.shape = (5,)`
"rank 1 array"

Don't use

`a = np.random.randn(5, 1) → a.shape = (5, 1)` column vector ✓

`a = np.random.randn(1, 5) → a.shape = (1, 5)` row vector ✓

```
assert(a.shape == (5,1)) ←  
a = a.reshape((5,1))
```

```
In [14]: %load_ext version_information  
%version_information numpy
```

Out[14]:

| Software | Version |
|------------------------------|--|
| Python | 3.6.6 64bit [GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.53)] |
| IPython | 7.0.1 |
| OS | Darwin 17.7.0 x86_64 i386 64bit |
| numpy | 1.15.1 |
| Sun Oct 14 19:41:16 2018 MDT | |