CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Machine Learning

Machine Learning

Lecture: Linear and Multivariate Regression

Ted Scully

Contents

Vectors

- The objective of vectorization is to remove or lessen the dependence on '**for loops**' within your code and to substitute matrix operations in their place.
- The consequence of this can significantly **speed up** your code.
- Modern CPUs and GPUs have **parallelization instructions**. By vectorising your code it allows your processor to take advantage of these instructions.
- Assume I had two row vectors W and X of equal length and I wanted to multiply each vector element-wise and add up the results. I could do this one of two ways.
- Please note below we use @ for .dot matrix multiplication. You could also use np.dot

```
result = 0
for num in range(0, len(W)):
    result += W[i] * X[i]
```

```
result =X@W.T
```

```
import numpy as np
import time

# create two column vectors
W = np.random.randn(10000000,1)
X = np.random.randn(10000000,1)

start = time.time()
result = W.T@X
print ("Answer:  ", result)
stop  = time.time()
durationV = 1000*(stop-start)
print ("Duration (Vector) is ", durationV)

start = time.time()
result = 0
for num in range(len(X)):
   result += X[num] * W[num]
stop  = time.time()
print ("Answer ", result)
durationF = 1000*(stop-start)

print ("Duration (For Loop) is ", durationF)
print ("Vector runtime as a fraction of for loop: ", durationV/durationF)
```

```
Answer:   [[-2531.60064132]]
Duration (Vector) is  6.934642791748047
Answer  [-2531.60064132]

Duration (For Loop) is  18434.74292755127
Vector runtime as a fraction of for loop:
0.0003761724705899759
```

Using Vectors in NumPy

- When using vectors in Python you need to make sure they you are using a true 2D array and not a **rank 1 (flat) array**. A failure to do so can cause a number of bugs in your code.
- The following is an example:

```
# create two rank one  (flat) arrays
import numpy as np

a = np.array( [12, 14, 15, 16] )

# doesn't behave like a row or column vector
print (a.shape)
print (a.T.shape)
```

```
(4,)
(4,)
```

Using Vectors in NumPy

- There are a number of simple ways around this. One is that you make sure the arrays are **2D NumPy** arrays when created.

```
# Notice we create 2 2D arrays
a = np.array( [ [12, 14, 15, 16] ] )

print (a.shape)
print (a.T.shape)
```

```
(1, 4)
(4, 1)
```

- Some functions will allow you to specify the shape of your NumPy array.

```
W = np.random.randn(10000000,1)
print (W.shape)

x = np.zeros((10000,1))
print (x.shape)
```

Using Vectors in NumPy

- Another simple solution to call **reshape** on the flat rank 1 arrays as follows:

```
# create two rank one array
a = np.array([12, 14, 15, 16])

# Here we alter a to be a row vector
a = a.reshape((1,4))

print (a.shape)
print (a.T.shape)
```

```
(1, 4)
(4, 1)
```

Broadcasting in NumPy

- The following broadcasting rules apply in **NumPY**

- If you have a **(m, n)** matrix and you perform a basic operation (+, *, /, -) between that matrix and a **(1, n)** matrix then NumPy will copy the (1, n) m times to create a matching size (m, n) matrix.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

Broadcasting in NumPy

- The following broadcasting rules apply in **NumPY**

- If you have a **(m, n)** matrix and you perform a basic operation (+, *, /, -) between that matrix and a **(m, 1)** matrix then NumPy will **copy** the (m, 1) n times to create a matching size (m, n) matrix.

$$\boxed{\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}} \qquad \boxed{\begin{matrix} 1 \\ 2 \end{matrix}}$$

$$\boxed{\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}} \qquad \boxed{\begin{matrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{matrix}}$$

Short Review of Matrices

Vector by Vector Multiplication (Dot product)

- We can multiply a vector **a** (1*n) by a vector **b** (n*1) by multiplying the first element of **a** by the first element of **b** and adding that to the product of the second element from a and b, and so on.

- The result will be a single scalar value. Notice if we use normal multiplication *, it will just perform element-wise multiplication and not add the products.
- Notice in the all the code examples that we never use flat rank 1 arrays.

```
import numpy as np

a = np.array([[15, 7, 3]])
b = np.array([[2, 3, 4]])

print ( a@(b.T) )
```

$$\begin{bmatrix} 15 & 7 & 3 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \quad \begin{bmatrix} 63 \end{bmatrix}$$

Short Review of Matrices

Matrix Vector Multiplication

- When multiplying a matrix by a vector **each row** of the matrix is multiplied by the vector.
- The number of columns in the in the matrix must match the number of rows in the vector.

- Matrix (m***n**) * Vector (**n***1) = Vector (m*1)

```
import numpy as np

M = np.array([[15,7],[4,3]])
y = np.array([[2,3]])

print ( M@( y.T ) )
```

[[15, 7], [4, 3]]	*	[2 3]	=	[51, 17]

[(15*2)+(7*3) ,
(2*4)+(3*3)]

Short Review of Matrices

Matrix - Matrix Multiplication

- You can view matrix by matrix multiplication as being multiple matrix by vector multiplication operations.
- Each **row** in turn from the **first matrix** is multiplied by **each column** in the **second matrix**.
- Therefore, the number of columns in the first matrix must match the number of rows in the second matrix.
- Matrix (m*$\mathbf{n}$) * Matrix ($\mathbf{n}$*p) = Vector (m*p)

```
import numpy as np

M = np.array([[15,7],[4,3]])
S = np.array([[2,1],[3,2]])

print (M@(S))
```

```
[[15  7]     [[2   1]     [[51 29]
 [4   3]]  *  [3   2]]  =  [17 10]]
```

```
[ (15*2)+(7*3)      (15*1)+(7*2)

  (4*2)+(3*3)       (4*1)+(3*2) ]
```

Short Review of Matrices

Matrix - Matrix Multiplication

- You can view matrix by matrix multiplication as being multiple matrix by vector multiplication operations.
- Each **row** in turn from the **first matrix** is multiplied by **each column** in the **second matrix**.
- Therefore, the number of columns in the first matrix must match the number of rows in the second matrix.
- Matrix (m*$\mathbf{n}$) * Matrix ($\mathbf{n}$*p) = Vector (m*p)

```
import numpy as np

M = np.array([[15,7],[4,3]])
S = np.array([[2,1],[3,2]])

print (M@(S))
```

$$\begin{matrix}[[\boxed{15\ \ 7]} \\ [4\ \ 3]]\end{matrix} \ * \ \begin{matrix}[[\boxed{2}\ \ 1] \\ [\boxed{3}\ \ 2]]\end{matrix} \ = \ \begin{matrix}[[51\ 29] \\ [17\ 10]]\end{matrix}$$

[$\boxed{(15*2)+(7*3)}$ (15*1)+(7*2)

(4*2)+(3*3) (4*1)+(3*2)]

Short Review of Matrices

Matrix - Matrix Multiplication

- You can view matrix by matrix multiplication as being multiple matrix by vector multiplication operations.
- Each **row** in turn from the **first matrix** is multiplied by **each column** in the **second matrix**.
- Therefore, the number of columns in the first matrix must match the number of rows in the second matrix.
- Matrix (m***n**) * Matrix (**n***p) = Vector (m*p)

```
import numpy as np

M = np.array([[15,7],[4,3]])
S = np.array([[2,1],[3,2]])

print (M@(S))
```

[[15 7]
[4 3]] * [[2 1]
[3 2]] = [[51 29]
[17 10]]

[(15*2)+(7*3) (15*1)+(7*2)

(4*2)+(3*3) (4*1)+(3*2)]

Short Review of Matrices

Matrix - Matrix Multiplication

- You can view matrix by matrix multiplication as being multiple matrix by vector multiplication operations.
- Each **row** in turn from the **first matrix** is multiplied by **each column** in the **second matrix**.
- Therefore, the number of columns in the first matrix must match the number of rows in the second matrix.
- Matrix (m***n**) * Matrix (**n***p) = Vector (m*p)

```
import numpy as np

M = np.array([[15,7],[4,3]])
S = np.array([[2,1],[3,2]])

print (M@(S))
```

```
[[15  7]      [[2   1]      [[51 29]
 [4   3]]  *   [3   2]]  =   [17 10]]
```

```
[ (15*2)+(7*3)      (15*1)+(7*2)

  (4*2)+(3*3)       (4*1)+(3*2) ]
```

Short Review of Matrices

Matrix - Matrix Multiplication

- You can view matrix by matrix multiplication as being multiple matrix by vector multiplication operations.
- Each **row** in turn from the **first matrix** is multiplied by **each column** in the **second matrix**.
- Therefore, the number of columns in the first matrix must match the number of rows in the second matrix.
- Matrix (m***n**) * Matrix (**n***p) = Vector (m*p)

```
import numpy as np

M = np.array([[15,7],[4,3]])
S = np.array([[2,1],[3,2]])

print (M@(S))
```

```
[[15  7]        [[2   1]      [[51 29]
 [4   3]]   *    [3   2]]  =   [17 10]]
```

[(15*2)+(7*3)   (15*1)+(7*2)

(4*2)+(3*3) $\boxed{(4*1)+(3*2)\ ]}$

Short Review of Matrices

Matrix - Matrix Multiplication

- You can view matrix by matrix multiplication as being multiple matrix by vector multiplication operations.
- Each **row** in turn from the **first matrix** is multiplied by **each column** in the **second matrix**.
- Therefore, the number of columns in the first matrix must match the number of rows in the second matrix.
- Matrix (m*n) * Matrix (n*p) = Vector (m*p)

```
import numpy as np

M = np.array([[15,7],[4,3]])
S = np.array([[2,1],[3,2]])

print (M@(S))
```

[[15 7]
[4 3]]
*
[[2 1]
[3 2]]
=
[[51 29]
[17 10]]

[(15*2)+(7*3)   (15*1)+(7*2)

(4*2)+(3*3)   (4*1)+(3*2)]

Using Matrices in Linear Regression

We can use matrices and vectors to help provide a more **succinct representation of our multiple linear regression** problem.

Let's assume that we want to predict the **rental price** of an office based on the **floor size** (x_1) and **proximity to city centre**(x_2). The model we are using is a simple multiple linear regression model.

$$h(x) = \lambda_1 x_1 + \lambda_2 x_2 + b$$

For example, let's assume I have developed a MLR model that has ***b* = -50.4 and $\boldsymbol{\lambda_1}$= 0.8 and = $\boldsymbol{\lambda_2}$= 0.2**. The purpose of the model is to predict the weekly rental amount of the office.

Using Matrices in Linear Regression

I could maintain a **data matrix X** and a single **parameter vector W** as follows:

```
[[  943, 2]
 [   1043, 3]
 [   678, 10]
 [   887, 1]]
```

```
[ 0.8, 0.2 ]
```

Therefore, we could represent our hypothesis
$h(x) = \lambda_1 x_1 + \lambda_2 x_2 + b$

And we could calculate the predicted rental price for each house as follows (using a normal for loop)

```
import numpy as np
X = np.array([[943, 2],[1043, 3], [678, 10], [887, 1]])
W = np.array([[0.8,0.2]])
b = -50.4

for house in X:
    predictedPrice = house[0]*W[0,0] + house[1]*W[0,1] + b
    print (predictedPrice)
```

Predicted house prices are as follows:

```
[[ 704.4]
 [ 784.6]
 [ 494. ]
 [ 659.4]]
```

Using Matrices in Linear Regression

I could maintain a **data matrix X** and a single **parameter vector W** as follows:

```
[[  943, 2]
 [   1043, 3]
 [   678, 10]
 [   887, 1]]
```

```
[ 0.8, 0.2 ]
```

In matrix form we could represent the hypothesis as
h(x) = X W^T + *b* *(where W is a row vector contains* λ_1 *and* λ_2 *)*

```
import numpy as np
X = np.array([[943, 2],[1043, 3], [678, 10], [887, 1]])
W = np.array([[0.8,0.2]])
b = -50.4

predictedPrices = X@(W.T) + b
print (predictedPrices)
```

Predicted house prices are as follows:

```
[[ 704.4]
 [ 784.6]
 [ 494. ]
 [ 659.4]]
```