§ Python Basics with Numpy

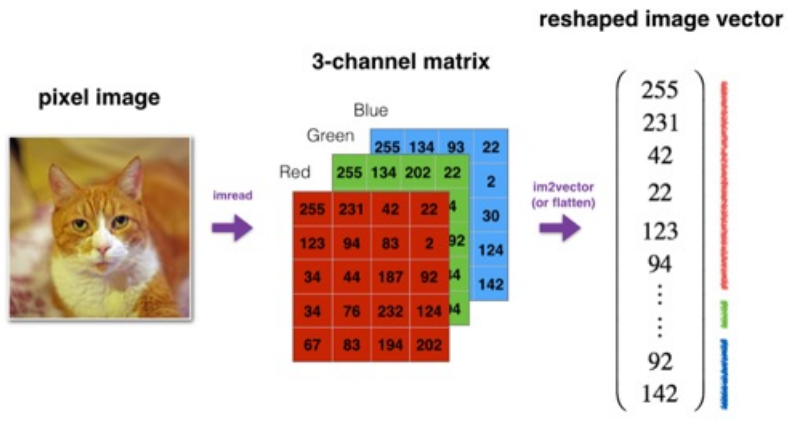
1. Building basic functions with numpy

(1) reshape

Two common numpy functions used in deep learning are np.shape and np.reshape().

- X.shape is used to get the shape (dimension) of a matrix/vector X.
- X.reshape(...) is used to reshape X into some other dimension.

For example, in computer science, an image is represented by a 3D array of shape $(length, height, depth = 3)$. However, when you read an image as the input of an algorithm you convert it to a vector of shape $(length * height * 3, 1)$. In other words, you "unroll", or reshape, the 3D array into a 1D vector.



Exercise: Implement `image2vector()` that takes an input of shape (length, height, 3) and returns a vector of shape (length*height*3, 1). For example, if you would like to reshape an array v of shape (a, b, c) into a vector of shape (a*b,c) you would do:

```
v = v.reshape((v.shape[0]*v.shape[1], v.shape[2])) # v.shape[0] = a ; v.shape[1] = b ; v.shape[2] = c
```

eg. image : a numpy array of shape (length, height, depth)

↓ v = v.reshape((v.shape[0] * v.shape[1] * v.shape[2], 1))

v : a vector of shape (length * height * depth, 1)

(2) Normalizing rows

Another common technique we use in Machine Learning and Deep Learning is to normalize our data. It often leads to a better performance because gradient descent converges faster after normalization. Here, by normalization we mean changing x to $\frac{x}{\|x\|}$ (dividing each row vector of x by its norm).

For example, if

$$x = \begin{bmatrix} 0 & 3 & 4 \\ 2 & 6 & 4 \end{bmatrix} \tag{3}$$

$\sqrt{0^2+3^2+4^2} = 5$

then

$$\|x\| = np.linalg.norm(x, axis = 1, keepdims = True) = \begin{bmatrix} 5 \\ \sqrt{56} \end{bmatrix} \tag{4}$$

and

$$x\_normalized = \frac{x}{\|x\|} = \begin{bmatrix} 0 & \frac{3}{5} & \frac{4}{5} \\ \frac{2}{\sqrt{56}} & \frac{6}{\sqrt{56}} & \frac{4}{\sqrt{56}} \end{bmatrix} \tag{5}$$

Note that you can divide matrices of different sizes and it works fine: this is called broadcasting and you're going to learn about it in part 5.

(3) Broadcasting

A very important concept to understand in numpy is "broadcasting". It is very useful for performing mathematical operations between arrays of different shapes. For the full details on broadcasting, you can read the official broadcasting documentation.

Exercise: Implement a softmax function using numpy. You can think of softmax as a normalizing function used when your algorithm needs to classify two or more classes. You will learn more about softmax in the second course of this specialization.

Instructions:

- for $x \in \mathbb{R}^{1\times n}$, $softmax(x) = softmax(\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}) = \begin{bmatrix} \frac{e^{x_1}}{\sum_j e^{x_j}} & \frac{e^{x_2}}{\sum_j e^{x_j}} & \dots & \frac{e^{x_n}}{\sum_j e^{x_j}} \end{bmatrix}$
- for a matrix $x \in \mathbb{R}^{m\times n}$, x_{ij} maps to the element in the i^{th} row and j^{th} column of x, thus we have:

$$softmax(x) = softmax\begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \cdots & x_{mn} \end{bmatrix} = \begin{bmatrix} \frac{e^{x_{11}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{12}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{13}}}{\sum_j e^{x_{1j}}} & \cdots & \frac{e^{x_{1n}}}{\sum_j e^{x_{1j}}} \\ \frac{e^{x_{21}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{22}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{23}}}{\sum_j e^{x_{2j}}} & \cdots & \frac{e^{x_{2n}}}{\sum_j e^{x_{2j}}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{e^{x_{m1}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m2}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m3}}}{\sum_j e^{x_{mj}}} & \cdots & \frac{e^{x_{mn}}}{\sum_j e^{x_{mj}}} \end{bmatrix} = \begin{pmatrix} softmax(\text{first row of x}) \\ softmax(\text{second row of x}) \\ \cdots \\ softmax(\text{last row of x}) \end{pmatrix}$$

2. Vectorization

(1) np.dot( )
- for 2-D arrays : matrix multiplication
- for 1-D arrays : inner product of vectors
- for N dimensions : dot(a,b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])

(2) loss function

①

- The loss is used to evaluate the performance of your model. The bigger your loss is, the more different your predictions ($\hat{y}$) are from the true values (y). In deep learning, you use optimization algorithms like Gradient Descent to train your model and to minimize the cost.
- L1 loss is defined as:

$$L_1(\hat{y}, y) = \sum_{i=0}^{m} |y^{(i)} - \hat{y}^{(i)}| \tag{6}$$

loss = np.sum(abs(yhat - y))

②

- L2 loss is defined as

$$L_2(\hat{y}, y) = \sum_{i=0}^{m} (y^{(i)} - \hat{y}^{(i)})^2 \tag{7}$$

loss = np.sum((yhat - y) * (yhat - y))