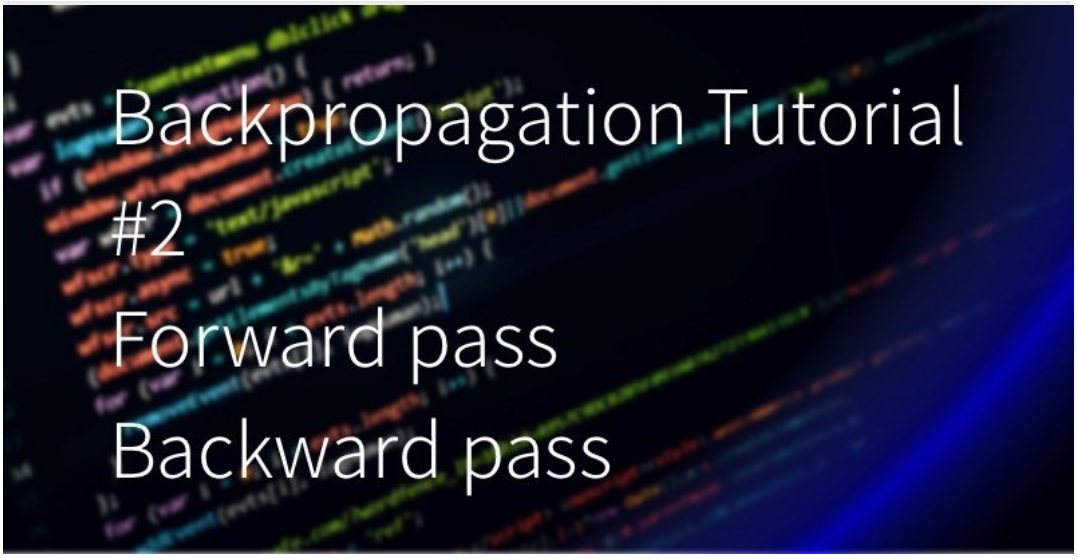


Machine Learning

Backpropagation(BP) 倒傳遞法 #2 貓貓分類器-2層類神經網路

2019-02-21 Andy Wang 0 Comments Backpropagation, Gradient Descent, Logistic Regression, Machine Learning, Neural Network, Optimization Algorithm

本篇會介紹在機器學習（machine learning）與深度學習（deep learning）領域裡很流行的倒傳遞法（Back Propagation/ Backpropagation, BP）的演算法流程與實作方法：正向傳遞（Forward pass）、反向傳遞（Backward pass）、邏輯回歸（Logistic

關注我們

Popular Posts

NumPy 1.14 教學 - #01 基礎, 建立陣列的方法

#05 (Ch9: 建構子、多型、封裝、覆載)

CentOS 7 安裝/設定SSH

NumPy 1.14 教學 - #03 基本操作(加減乘除、矩陣乘法、取代)

NumPy 1.14 教學 – #07 用陣

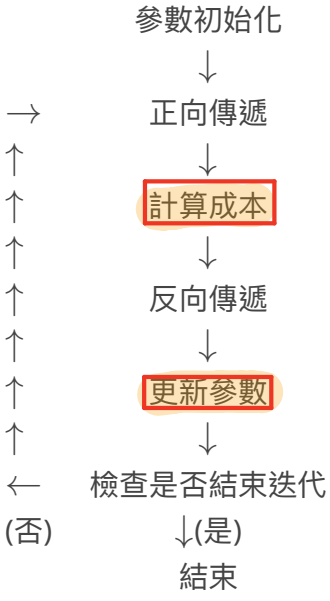
> regression）。除此之外，本篇會用簡易的2層類神經網路建立一個『貓貓分類器』。

先來GitHub下載這個範例吧！邊執行邊看文章比較好理解 😀
如果有需要更詳細的原理或是有什麼寫錯的，歡迎在文末留言哦！

如果你覺得還是不太懂**推導過程**可以先來讀這篇：Backpropagation(BP) 倒傳遞法 #1 工作原理與說明；你想要知道該如何優化**多層類神經網路**可以讀這篇：Backpropagation(BP) 倒傳遞法 #3 貓貓分類器–N層類神經網路

演算法流程

下圖（1）是演算法流程，本文章也會依照這個流程介紹演算法的實作方法。其中『正向傳遞』到『檢查是否結束迭代』之間的4個步驟就是迭代迴圈的所在。



圖（1）：倒傳遞法演算法流程

列當索引取值(Indexing with array of indices)

Python3 教學、筆記

Python3 教學 #04 (Ch6~Ch8: Try-catch 錯誤處理)

Python3 教學 #01 (Ch1~Ch4: 環境說明、運算子、流程控制)

CentOS 7 安裝 Apache 2.4 (httpd)

如何申請免費SSL？又要怎麼安裝SSL？(以SSL For Free - Let's Encrypt為例)

前置作業

載入資料集、函式庫

```python
import time
import numpy as np
import h5py
import matplotlib.pyplot as plt
import scipy
from PIL import Image
from scipy import ndimage
from dnn_app_utils_v3 import *

%matplotlib inline

plt.rcParams['figure.figsize'] = (5.0, 4.0) # set default size of p
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

%load_ext autoreload

%autoreload 2


np.random.seed(1)
```

資料集

Coursera課程中所使用的資料集為64×64的彩色圖片，一共有209張圖片作為訓練資料集、50張圖片為測試資料集。我們可以透過下列程式觀察。

```python
# Load dataset.
train_x_orig, train_y, test_x_orig, test_y, classes = load_data()
# Example of a picture
index = 166
plt.figure(num=1, figsize=(3,3))
plt.imshow(train_x_orig[index])
print ("y = " + str(train_y[0,index]) + ". It's a " + classes[train

# Explore your dataset
m_train = train_x_orig.shape[0]
num_px = train_x_orig.shape[1]
m_test = test_x_orig.shape[0]
print ("train_x_orig.shape: " + str(train_x_orig.shape))
print("")
print ("Number of training examples: " + str(m_train))
print ("Number of testing examples: " + str(m_test))
print ("Each image is: (" + str(num_px) + ", " + str(num_px) + ", 3
print("")
print ("train_x_orig shape: " + str(train_x_orig.shape))
print ("train_y shape: " + str(train_y.shape))
print ("test_x_orig shape: " + str(test_x_orig.shape))
```

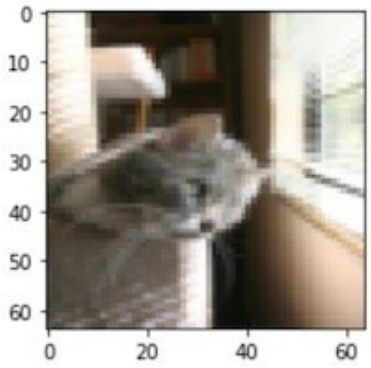
```
print ("test_y shape: " + str(test_y.shape))
```

Output

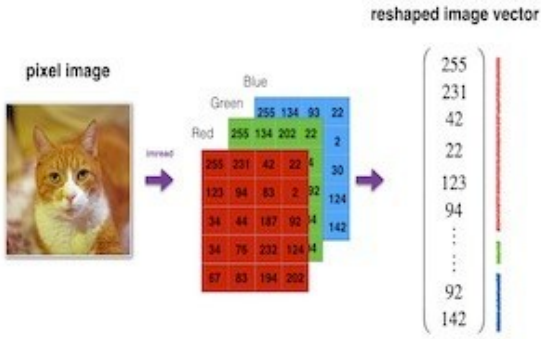
```
y = 1. It's a cat picture.
train_x_orig.shape: (209, 64, 64, 3)

Number of training examples: 209
Number of testing examples: 50
Each image is: (64, 64, 3)

train_x_orig shape: (209, 64, 64, 3)
train_y shape: (1, 209)
test_x_orig shape: (50, 64, 64, 3)
test_y shape: (1, 50)
```



這段程式可以發現 train_x_orig的維度排列依序是 每一張圖片、長度、寬度、RGB。接下來，我們要對這些圖片資料做前處理，先將每一張圖片壓縮成$12288 - by - 1$的向量。為什麼是12288？因為長x寬xRGB$= 64 \times 64 \times 3 = 12288$，如圖（2）所示。



圖（2）：圖片轉向量過程示意圖

如下方程式，首先將train_x_orig改變形狀成12288×209，為了加強模型效用與收斂速度再將所有的剛才轉換好的209張圖片內的數值做Normalization運算（資料集中每筆數據 ÷ 資料集中的最大值），這讓所有資料都會介於0到1之間。相同的做法也需要再對test_x_orig做一次。如此一來，我們就算是把資料集完成前

處理了。

```python
# Preprocess input data(images)
# Reshape the training and test examples
train_x_flatten = train_x_orig.reshape(train_x_orig.shape[0], -1).T
test_x_flatten = test_x_orig.reshape(test_x_orig.shape[0], -1).T

# Normalization data to have feature values between 0 and 1.
train_x = train_x_flatten/255.
test_x = test_x_flatten/255.

print ("train_x's shape: " + str(train_x.shape))
print ("test_x's shape: " + str(test_x.shape))
```

Output

```
train_x's shape: (12288, 209)
test_x's shape: (12288, 50)
```

架構

在開始實作模型所需程式之前，先說明本文的重點2層的邏輯回歸模型架構。下圖（3）是以單張圖片輸入2層類神經網路、2層網路皆僅以1顆神經元的示意圖，而輸入圖片將會是經過上述程序向量化後的圖片。

然而，此架構圖是引用自Coursera課程中的第3週所使用的架構圖，但是對於本文的目標來說還需要在最後加上一個成本函數，這部分會在本節『各層架構』中說明。

記號說明

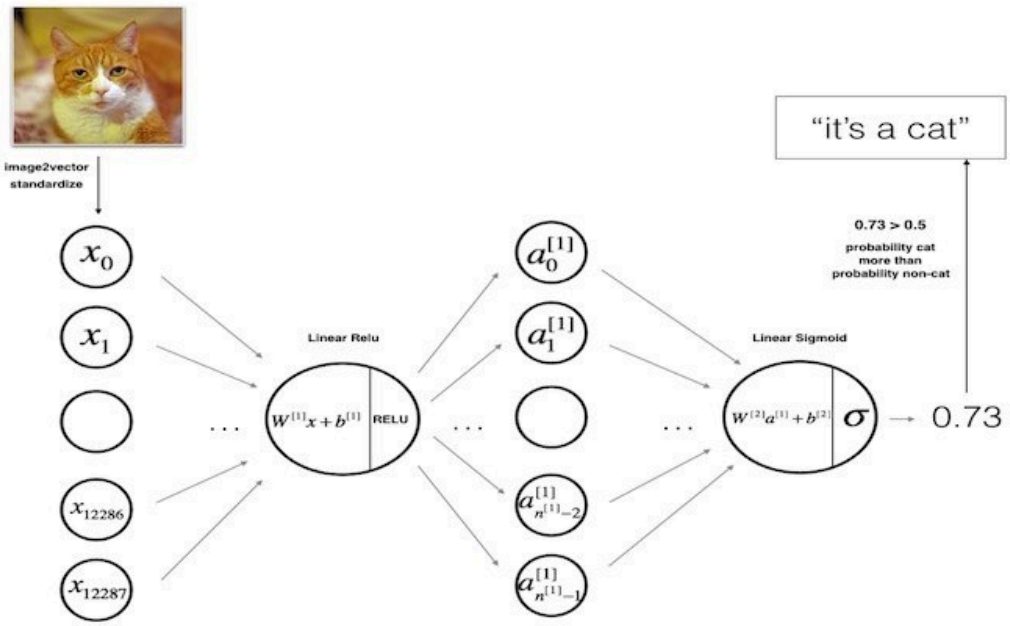
- 上標$[1]$代表第1層類神經網路
- $n^{[1]}$表示的是模型中第1層類神經網路的神經元輸出的數量
- $a_5^{[1]}$代表第1層類神經網路神經元的第$5+1$筆輸出（因為神經元輸出編號是從0開始，所以要$+1$，又因為神經元只有1顆所以沒有標記號）
- $a^{[2]}$表示類神經網路中整個第2層的所有輸出
- 在圖中還可以發現有個大圓上方標著Linear Relu表示該層網路所採用的激勵函數（activation function）為ReLU，然而本架構所使用的激勵函數分別為第1層為ReLU、第2層為Sigmoid。
- 各層神經元內部計算則是這樣，第1層的神經元計算以公式

(1) 、 (2) 所示

$$Z^{[1]} = w^{[1]}x + b^{[1]}$$	(1)
$$a^{[1]} = ReLU(Z^{[1]})$$	(2)

各層架構

第1層類神經網路中每顆神經元會有$n^{[1]}$個神經元輸出，分別是$a_0^{[1]} \cdots a_{(n^{[1]}-1)}$。
第2層類神經網路中每顆神經元會有1個神經元輸出。
最後，這張圖還沒加上成本函數，應該要把0.73改成$a_0^{[2]}$然後畫一個箭頭指向成本函數$L(y, a^{[2]})$代表輸入成本函數L，這邊的$a^{[2]}$少了下標的原因是指整個第2層的輸出。就像這樣：
$a_0^{[2]} \rightarrow L(y, a^{[2]})$



圖（3）：2層邏輯回歸模型架構

成本函數

圖（3）第2層神經元計算出來的0.73是輸入成本函數（Cost function）計算之前的數值，僅能算是$a_0^{[2]}$。那成本函數是什麼？成本函數就是用來判斷這些參數對於這個模型好壞的依據。本文

所使用的成本函數如公式（3）所示，本公式裡有個變數m指的是樣本數，例如：訓練階段有209張圖片則$m = 209$、測試階段有50張圖片則$m = 50$。如果看不懂，也可以先忽略m的存在。
（此處成本函數的$\log$是以e為底的Natural log（$\ln$），使用Natural log的原因很簡單，因為$\ln$的微分規則比較簡單，而這也算是使用Backpropagation的慣例，很多時候發論文的作者也都不太提了）

$$L(y, a^{[2]}) = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right) \quad (3)$$

參數初始化

初始化參數有兩個規則：w要用常態分佈亂數初始化、b初始值都是0。簡單吧！
（這邊使用的np是Numpy的別名）
常態分佈亂數：np.random.randn(維度)
零矩陣：np.zeros([維度])

下列程式傳入三個參數（n_x、n_h、n_y），用途與原理解釋如下：
n_x：要輸入每一張圖片的維度（$64 \times 64 \times 3$）
n_h：第1層的神經元數量（我們要設計成這部分能夠隨時自訂）
n_y：第二層神經元的數量（第2層固定為1顆神經元）

原理是什麼？為何這樣設計？我們可以先看一下圖（4）然後再來看表（1）、表（2）。其設計原理就是要符合矩陣運算時需要注意的形狀變化與計算規則，當然最重要的是模型需求（第1層的神經元數量要可以隨時更改）。可以先從表（1）看到形狀設定值。計算過程可以參考表（2），此表格包括第1、2層類神經網路的操作與每一個步驟對應到的變數形狀，然而$b^{[1]}$、$b^{[1]}$沒在計算過程中寫上形狀的原因是參數b的形狀基本上不會影響輸出結果的形狀。

表（1）：初始化變數形狀

參數/變數名稱	形狀
train_x（x）	12288×209
W1（$w^{[1]}$）	n_h×12288
b1（$b^{[1]}$）	n_h×1
W1（$w^{[2]}$）	n_y×n_h
b1（$b^{[2]}$）	n_y×1

表（2）：計算過程之形狀變化表

Layer-1	步驟1	$Z^{[1]} = w^{[1]}x + b^{[1]}$
	步驟1形狀	$[\text{n\_h}\times 12288] \cdot [12288 \times 209] + b^{[1]} \Rightarrow [\text{n\_h}\times 209]$
	步驟2	$a^{[1]} = ReLU(Z^{[1]})$
	步驟2形狀	$[\text{n\_h}\times 209]$（ReLU輸出形狀不變）
Layer-2	步驟1	$Z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
	步驟1形狀	$[\text{n\_y}\times \text{n\_h}]\cdot[\text{n\_h}\times 209] + b^{[2]} \Rightarrow [\text{n\_y}\times 209]$
	步驟2	$a^{[2]} = Sigmoid(Z^{[2]})$
	步驟2形狀	$[\text{n\_y}\times 209]$（Sigmoid輸出形狀不變）

```python
def initialize_parameters(n_x, n_h, n_y):
    """
    Argument:
    n_x -- size of the input layer
    n_h -- size of the hidden layer
    n_y -- size of the output layer

    Returns:
    parameters -- python dictionary containing your parameters:
                    W1 -- weight matrix of shape (n_h, n_x)
                    b1 -- bias vector of shape (n_h, 1)
                    W2 -- weight matrix of shape (n_y, n_h)
                    b2 -- bias vector of shape (n_y, 1)
    """

    np.random.seed(1)
    W1 = np.random.randn(n_h, n_x)*0.01
    b1 = np.zeros([n_h, 1])
    W2 = np.random.randn(n_y, n_h)*0.01
    b2 = np.zeros([n_y, 1])

    parameters = {"W1": W1, "b1": b1, "W2": W2, "b2": b2}

    return parameters
```

正向傳遞

正向傳遞（Forward pass）就是從圖片輸入模型開始到計算出成本的過程，但是我們把計算成本的函數獨立出來，所以此函數裡面沒有計算成本的部分。forwardpass這個函數除了會回傳第2層的計算結果（$a^{[2]}$）之外，還會回傳一個變數cache，這個變數的用途是反向傳遞計算過程的參數。

Python

```python
def forwardpass(X, parameters):
    """
    Argument:
    X -- input data of size (n_x, m)
    parameters -- python dictionary containing your parameters (out

    Returns:
    A2 -- The sigmoid output of the second activation
    cache -- a dictionary containing "Z1", "A1", "Z2" and "A2"
    """

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1, X) + b1
    A1 = np.maximum(0,Z1)        # ReLU
    Z2 = np.dot(W2, A1) + b2
    A2 = 1/(1+np.exp(-Z2))       # Sigmoid

    cache = {"Z1": Z1, "A1": A1, "Z2": Z2, "A2": A2}
    return A2, cache
```

計算成本

公式（3）這是本次Logistic Regression模型的成本函數，這公式裡面有個討人厭的$\sum$，和一個意味不明的變數m。（公式（3）和公式（1）是一樣的公式，為方便閱讀再複製到這）其實m是指輸入照片的數量，若是訓練階段m就是209，測試階段的話m則是50。公式（4）是為把$\sum$去除掉的作法。其中公式（3）、公式（4）都可以看見變數m，要將成本通除以m的原因是要取得平均成本。最後，y指的又是什麼？y是解答，train_y的形狀就是一個1×209的向量、test_y的形狀則是1×50的向量，數值的話就只有1和0兩種，1表示這張圖片是貓，0代表不是貓。

$$L(y, a^{[2]}) = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right) \quad (3)$$

$$L(y, a^{[2]}) = (ya^{[2]T} - (1-y)(\log(1-a^{[2]}))^T)/m \quad (4)$$

```python
def compute_cost(A2, Y, parameters):
    m = Y.shape[1]   # number of calss

    #cost = - np.sum(np.multiply(np.log(A2), Y) + np.multiply(1-Y,
    cost = (1./m) * (-np.dot(Y,np.log(A2).T) - np.dot(1-Y, np.log(1

    cost = np.squeeze(cost)  # makes sure cost is the dimension we
    return cost
```

反向傳遞

反向傳遞（Backward pass），這是整個倒傳遞法最困難的部分！

但是，我們可以參考上一篇中連鎖率的概念最終計算出這幾個參

數：$\frac{\partial L}{\partial w^{[1]}}$ 、$\frac{\partial L}{\partial b^{[1]}}$ 、$\frac{\partial L}{\partial w^{[2]}}$ 、$\frac{\partial L}{\partial b^{[2]}}$

概念就是$a^{[2]}$對模型成本偏微分→對激勵函數（Sigmoid）偏微分→對線性函數偏微分（$w^{[2]}a^{[1]}+b^{[2]}$）→對激勵函數（ReLU）偏微分→ $\cdots$

$$\frac{\partial L}{\partial a^{[2]}} = -(\frac{y}{a^{[2]}} - \frac{1-y}{1-a^{[2]}})$$	(5)
$$\frac{\partial L}{\partial Z^{[2]}} = \frac{\partial L}{\partial a^{[2]}}(\frac{1}{1+e^{(-Z^{[2]})}})(1-\frac{1}{1+e^{(-Z^{[2]})}})$$	(6)
$$\frac{\partial L}{\partial w^{[2]}} = (\frac{\partial L}{\partial Z^{[2]}}a^{[1]T})/m$$	(7)
$$\frac{\partial L}{\partial b^{[2]}} = (\frac{\partial L}{\partial Z^{[2]}}a^{[1]T})/m$$	(8)
$$\frac{\partial L}{\partial a^{[1]}} = w^{[2]T}\frac{\partial L}{\partial Z^{[2]}}$$	(9)
$$\frac{\partial L}{\partial Z^{[1]}} =$$ $\frac{\partial L}{\partial a^{[1]}}$對應至$Z^{[1]}$相同位置的數值$\leq 0$的都改成0	(10)
$$\frac{\partial L}{\partial w^{[1]}} = \frac{\partial L}{\partial Z^{[1]}}x^T/m$$	(11)
$$\frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial Z^{[1]}}/m$$	(12)

Python

```python
def backwardpass(parameters, cache, X, Y):
    """
    Implement the backward propagation using the instructions above

    Arguments:
    parameters -- python dictionary containing our parameters
    cache -- a dictionary containing "Z1", "A1", "Z2" and "A2".
```

```
    X -- input data of shape (2, number of examples)
    Y -- "true" labels vector of shape (1, number of examples)

    Returns:
    grads -- python dictionary containing your gradients with respe
    """

    m = X.shape[1]

    W1 = parameters["W1"]
    W2 = parameters["W2"]
    A1 = cache["A1"]
    A2 = cache["A2"]
    Z1 = cache["Z1"]
    Z2 = cache["Z2"]

    dA2 = - (np.divide(Y, A2) - np.divide(1 - Y, 1 - A2))

    temp_s = 1/(1+np.exp(-Z2))
    dZ2 = dA2 * temp_s * (1-temp_s)        # Sigmoid (back propagat

    dW2 = 1/m * np.dot(dZ2, A1.T)
    db2 = 1/m * np.sum(dZ2, axis=1, keepdims=True)
    dA1 = np.dot(W2.T,dZ2)

    # ReLU (back propagation)
    dZ1 = np.array(dA1, copy=True) # just converting dz to a corre
    dZ1[Z1 <= 0] = 0   # When z <= 0, you should set dz to 0 as we

    dW1 = 1/m * np.dot(dZ1, X.T)
    db1 = 1/m * np.sum(dZ1, axis=1, keepdims=True)

    grads = {"dW1": dW1, "db1": db1, "dW2": dW2, "db2": db2}
    return grads
```

更新參數

參數更新只有一個原則，就是現在的參數減掉Backward pass計算出來的梯度值乘上學習速率（Learning rate）。

$$w^{[1]} = w^{[1]} - \alpha(\frac{\partial L}{\partial w^{[1]}}) \quad (13)$$

$$b^{[1]} = b^{[1]} - \alpha(\frac{\partial L}{\partial b^{[1]}}) \quad (14)$$

$$w^{[2]} = w^{[2]} - \alpha(\frac{\partial L}{\partial w^{[2]}}) \quad (15)$$

$$b^{[2]} = b^{[2]} - \alpha(\frac{\partial L}{\partial b^{[2]}}) \tag{16}$$

```python
def update_parameters(parameters, grads, learning_rate = 1.2):
    """
    Updates parameters using the gradient descent update rule giver

    Arguments:
    parameters -- python dictionary containing your parameters
    grads -- python dictionary containing your gradients

    Returns:
    parameters -- python dictionary containing your updated paramet
    """

    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    dW1 = grads["dW1"]
    db1 = grads["db1"]
    dW2 = grads["dW2"]
    db2 = grads["db2"]

    W1 = W1 - learning_rate*dW1
    b1 = b1 - learning_rate*db1
    W2 = W2 - learning_rate*dW2
    b2 = b2 - learning_rate*db2

    parameters = {"W1": W1, "b1": b1, "W2": W2, "b2": b2}

    return parameters
```

主程式

主程式最主要就是要可以讓人方便操作，透過簡單的給予幾個重要參數就可以開始訓練參數。

nn_model的輸入值X就是被向量化的圖片，每一張圖片的維度是12288×1，以訓練階段來說X一共有209張圖片，而Y就是這209張圖片相對應的分類（class）數值則分為貓或非貓（1、0）。

```python
def nn_model(X, Y, n_h, num_iterations = 5000, learning_rate=0.08,
    """
    Arguments:
    X -- dataset of shape (2, number of examples)
```

```
    Y -- labels of shape (1, number of examples)
    n_h -- size of the hidden layer
    num_iterations -- Number of iterations in gradient descent loop
    print_cost -- if True, print the cost every 1000 iterations

    Returns:
    parameters -- parameters learnt by the model. They can then be
    """
    costs = []

    np.random.seed(1)
    n_x = X.shape[0]
    n_y = Y.shape[0]

    # Initialize W1, b1, W2, b2
    parameters = initialize_parameters(n_x, n_h, n_y)
    #print("W1.shape: " + str(parameters["W1"].shape))
    #print("W2.shape: " + str(parameters["W2"].shape))
    for i in range(0, num_iterations):
        A2, cache = forwardpass(X, parameters)
        #print("Z1.shape: " + str(cache["Z1"].shape))
        #print("A1.shape: " + str(cache["A1"].shape))
        #print("Z2.shape: " + str(cache["Z2"].shape))
        #print("A2.shape: " + str(cache["A2"].shape))
        cost = compute_cost(A2, Y, parameters)
        grads = backwardpass(parameters, cache, X, Y)
        parameters = update_parameters(parameters, grads, learning_

        if i % 500 == 0:
            costs.append(cost)
            if print_cost:
                print("Cost after iteration {}: {}".format(i, cost)
    # The latest iteration.
    print("Cost after iteration {}: {}".format(i, cost))
    costs.append(cost)

    plt.figure(num=1, figsize=(8,5))
    plt.semilogy(costs)
    plt.xlabel("Iterations")
    plt.ylabel("Cost")
    plt.title("Learning Rate = " + str(learning_rate))
    plt.show()

    return parameters
```

判斷準確度

判斷準確度的函數是要利用正向傳遞（Forward pass）協助完成，做法是把訓練好的參數（parameters）和向量化的圖片（每張圖的維度是12288×1）丟進forwardpass，回傳值則是probas（$a^{[2]}$）和cache（cache在預測精準度時沒有用到）。判斷規則是probas> 0.5判斷為貓，反之為非貓～

Python

```
def predict(X, y, parameters):
    """
    This function is used to predict the results of a  L-layer neur
```

```
    Arguments:
    X -- data set of examples you would like to label
    parameters -- parameters of the trained model

    Returns:
    p -- predictions for the given dataset X
    """

    m = X.shape[1]
    n = len(parameters) // 2 # number of layers in the neural netw
    p = np.zeros((1,m))

    # Forward propagation
    probas, caches = forwardpass(X, parameters)


    # convert probas to 0/1 predictions
    for i in range(0, probas.shape[1]):
        if probas[0,i] > 0.5:
            p[0,i] = 1
        else:
            p[0,i] = 0

    #print results
    #print ("predictions: " + str(p))
    #print ("true labels: " + str(y))
    print("Accuracy: "  + str(np.sum((p == y)/m)))

    return p
```

怎麼使用

訓練模型

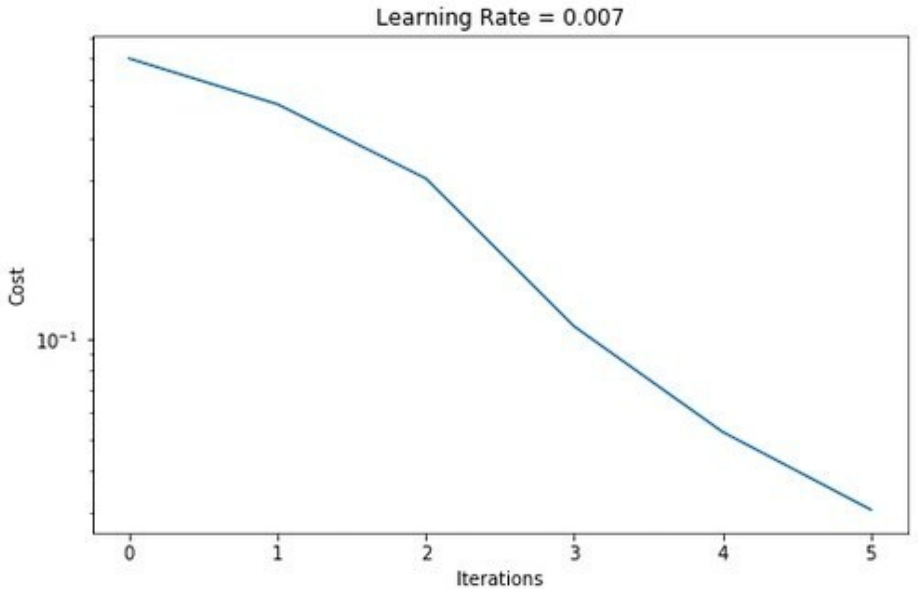
模型怎麼訓練啊？

直接傳入向量化的圖片train_x、解答train_y、第1層神經網路的神經元數量12、迭代次數num_iterations、學習速率learning_rate、是否要每隔500代印一次目前成本print_cost，然後用一個變數來收nn_model訓練結束所回傳的參數，本範例用的是parameters。

```python
parameters = nn_model(train_x, train_y, 12, num_iterations=2500, le
```

Output

```
Cost after iteration 0: 0.6933973875299138
Cost after iteration 500: 0.5054817305127275
Cost after iteration 1000: 0.3024003130312214
Cost after iteration 1500: 0.10870519536443567
Cost after iteration 2000: 0.05241476625572783
Cost after iteration 2499: 0.030590797593466793
```



準確度判斷

判斷準確度的工作就交給函數predict囉～這應該不需要多解釋了🙂

```python
print("Training accuracy:")
predictions_train = predict(train_x, train_y, parameters)
print("Testing accuracy:")
predictions_test = predict(test_x, test_y, parameters)
```

Output

```
Training accuracy:
Accuracy: 0.9999999999999998
Testing accuracy:
Accuracy: 0.74
```

自己找一張圖測試

我們可以丟一張自己準備的圖片進模型使用剛才訓練完的參數做測試，下列程式第9行有個變數num_px是在最一開始查看『資料集』時存下來的變數，意思是訓練階段與測試階段的圖片長、寬。

```python
## START CODE HERE ##
# cat.jpg  my_image.jpg  people.jpeg
my_image = "cat.jpg" # change this to the name of your image file
my_label_y = [1] # the true class of your image (1 -> cat, 0 -> no
```

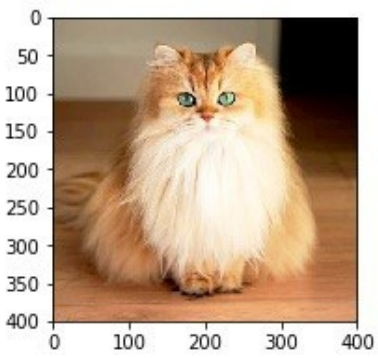
```
## END CODE HERE ##

fname = "images/" + my_image
image = np.array(ndimage.imread(fname, flatten=False))
my_image = scipy.misc.imresize(image, size=(num_px,num_px)).reshape
my_image = my_image/255.
my_predicted_image = predict(my_image, my_label_y, parameters)

plt.figure(num=1, figsize=(3,3))
plt.imshow(image)
print ("y = " + str(np.squeeze(my_predicted_image)) + ", your L-lay
```

Output

```
Accuracy: 1.0
y = 1.0, your L-layer model predicts a "cat" picture.
```



References

1. Andrew Ng — Neural Networks & Deep Learning in Coursera
2. (paper) Learning representations by back-propagation errors

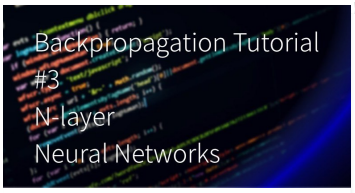
← Backpropagation(BP) 倒傳遞法 #1 工作原理與說明

Backpropagation(BP) 倒傳遞法 #3 貓貓分類器-N層類神經網路 →

Andy Wang

站在巨人的肩膀上仍須戰戰兢兢！

You May Also Like



Backpropagation (BP) 倒傳遞法 #3 貓貓分類器–N層類神經網路

2019–04–25 0

Backpropagation (BP) 倒傳遞法 #1 工作原理與說明

2019–02–21 0

使用 Ubuntu 作為深度學習/機器學習/人工智慧之 平台

2018–03–25 0

發表迴響

你的電子郵件位址並不會被公開。 必要欄位標記為 *

迴響

名稱 *

電子郵件 *

個人網站

☐ 用電子郵件通知我後續的迴響。

☐ 新文章使用電子郵件通知我。

張貼迴響

Tags

Anaconda (2) Apache (8) Backpropagation (3) BeautifulSoup (1) Bitbucket (2) CentOS (36) Certbot (2) Container (1) cron (2) CUDA (5) Docker (1) Drupal (1) FreeNAS (1) FTP Server (2) Git (2) Gradient Descent (3) Homebrew (3) htaccess (1) https (5) iptables (1) Linode (2) Linux (25) Logistic Regression (2) lxml (1) Mac (7) Machine Learning (4) mount (2) MySQL (7) Neural Network (2) NextCloud (1) NumPy (11) Optimization Algorithm (3) php (2) phpMyAdmin (4) Python (15) Requests (1) SELinux (1) SMB (1) SSH (3) SSL (9) Sublime (2) Tensorflow (3) Ubuntu (8) VPS (1) yum (3)

About

Brilliant Code.net

在這個傳統資訊科技與人工智慧即將盛行的年代提供一個學習機器學習領域技術知識的良好機會，希望你我一起學習成長迎向這個機會。

若有任何疑問，歡迎來信。
Email: polun.wang@gmail.com

Statistics

線上使用者: 4

今日瀏覽次數: 571

最後 30 天瀏覽: 25,339

總瀏覽次數: 219,242