Single Neuron Image Classifier

Shaurya Prakash
Ph.D Second Semester
Supervisor : Dr.M.K.Singh

Banaras Hindu University ,Varanasi

2022

A labeled dataset of images :

Images To Regression

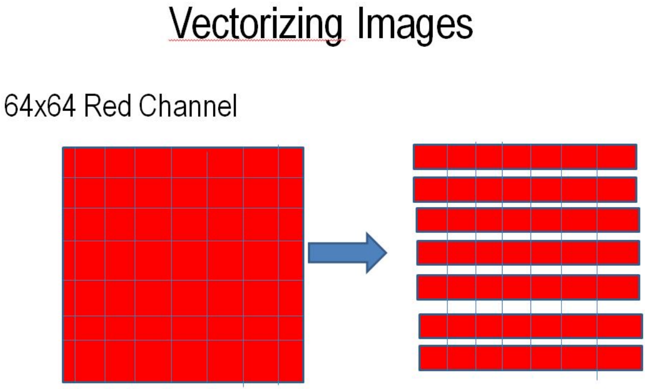
Image : Label

A cat image

$\{x_1, x_2 \ldots\ldots x_n\}$: $\{ y_1$:cat , y_0:not a cat image$\}$

After this we can apply regression / ANN to fit a linear separator line.

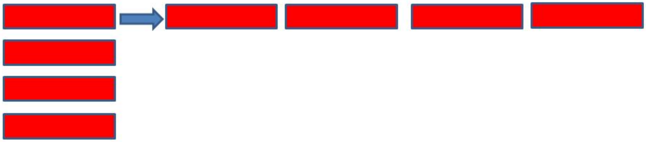
How to vectorize a single channel : taking slices



How to vectorize a single channel : concatenating slices

Vectorizing Images

Vectorizing only red-channel



64 x 64 $\rightarrow$ 4096 x 1

How to vectorize a RGB channel : concatenating RGB slices

Vectorizing Images

Concatenating all vectorized channels

64 x 64 x 3 $\rightarrow$ 12288 x 1

Image = X : $[x_1, x_2, x_3 \ldots\ldots\ldots\ldots x_{12288}]$ Y :{0,1}

Single Input Vector

A single input vector is represented by :

$$x_{12288,1} = \begin{bmatrix} x_{1,1} \\ x_{2,1} \\ \vdots \\ x_{12288,1} \end{bmatrix}$$

Single Input Weight Vector

A single weight vector is represented by :

$$w_{12288,1} = \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{12288,1} \end{bmatrix}$$

Evaluation of aggregate sum

$$Z = W^\top \times X + b$$

$$\frac{\partial Z}{\partial w} = X^\top$$

$$\frac{\partial Z}{\partial b} = 1$$

Activation Function : Sigmoid

The input vector z is fed to sigmoid function to evaluate activation value :

$$a(z) = \frac{1}{1 + e^{-z}}$$

Activation Function : Sigmoid

The input vector z is fed to sigmoid function to evaluate activation value :

$$\frac{\partial a(z)}{\partial z} = \frac{-e^{-z}}{(1+e^{-z})^2}$$

$$\frac{\partial a(z)}{\partial z} = \frac{1}{1+e^{-z}} \times \frac{(1-1-e^{-z})}{1+e^{-z}}$$

$$\frac{\partial a(z)}{\partial z} = \frac{1}{1+e^{-z}} \times \frac{(1-(1+e^{-z})}{1+e^{-z}}$$

$$\frac{\partial a(z)}{\partial z} = a(z) \times (1-a(z))$$

$$\frac{\partial a}{\partial z} = a \times (1-a)$$

Gradient Descent : General Formulae

Learning Rule for weight and bias are :

$$W_{new}^{[L]} = W_{old}^{[L]} - \alpha \frac{\partial J}{\partial w^{[L]}}$$

$$b_{new}^{[L]} = b_{old}^{[L]} - \alpha \frac{\partial J}{\partial b^{[L]}}$$

where ,

α is learning rate , should be kept low to avoid instability.

$\frac{\partial J}{\partial w^{[L]}}$ is gradient , i.e. direction of maximum increase in weights.

$-\frac{\partial J}{\partial w^{[L]}}$ is direction of maximum decrease in weights.

Gradient Descent : Adding apple and oranges ?

Learning Rule for weight and bias are :

$$W_{new}^{[L]} = W_{old}^{[L]} - \alpha \frac{\partial J}{\partial w^{[L]}}$$

subtituting following :

$$\frac{\partial J}{\partial w^{[L]}} = (a - y) \times x$$

we get interesting result

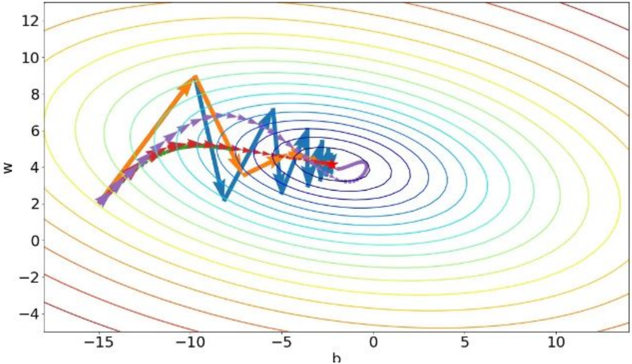
$$W_{new}^{[L]} = W_{old}^{[L]} - \alpha(a - y) \times x$$

on rearranging :it becomes PERCEPTRON LEARNING RULE

$$W_{new}^{[L]} = W_{old}^{[L]} + \alpha(y - a) \times x$$

Gradient Descent : Effect of Learning Rate

Large gradient may lead to instability.
Small gradient slows down the speed of learning.



Full Dataset as Matrix

The input vector to a model is represented by :

$$X_{12288,208} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,208} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,208} \\ \vdots & \vdots & \ddots & \vdots \\ x_{12288,1} & x_{12288,2} & \cdots & x_{12288,208} \end{bmatrix}$$

Evaluation of aggregate sum

We know that $Z = W^\top \times X + b$, in matrix form it is represented as follows :

$$\begin{bmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{12288,1} \end{bmatrix}^\top \times \begin{bmatrix} x_{1,1} & \cdots & x_{1,208} \\ x_{2,1} & \cdots & x_{2,208} \\ \vdots & \vdots & \vdots \\ x_{12288,1} & \cdots & x_{12288,208} \end{bmatrix} + b = \begin{bmatrix} z_1 & .. & z_{208} \end{bmatrix}$$

$$z_1 = w_{1,1}x_{1,1} + w_{2,1}x_{2,1} + \cdots + w_{12288,1} \times x_{12288,1} + b = \sum_{k=1}^{12288} w_{ik}x_{kj}$$

Here z_1 is a scalar value , evaluated by taking dot product of weight vector $w_{12288,1}$ and input vector $x_{12288,1}$.

Binary Cross Entropy Function : Minimizing Surprise element

$$J(w,b) = -\frac{1}{208} \times \sum_{k=1}^{208} \times \left[y \log_2 a + (1-y) \log_2 (1-a) \right]$$

When Y=0 : Not a cat

$$J(w,b) = -\left[\log_2 (1-a) \right]$$

When Y=1 : Cat

$$J(w,b) = -\left[\log_2 (a) \right]$$

Y = label(0/1) , a = sigmoid function output i.e. prediction

Binary Cross Entropy Function :

When Y=1 :Cat

$$J(w,b) = -\left[\log_2(a)\right]$$

$-\left[\log_2(a)\right]$ is measure of element of surprise.

Low value of 'a' means high measure of surprise .

We want to develop a model which should not have surprise element in prediction.

If we want to see a cat , we get a cat : LOW SURPRISE

If we want to see a cat , we see a dog : HIGH SURPRISE

Hence , we want to minimize the surprise element by minimizing cost function , which is measure of surprise.

Binary Cross Entropy Function :

information(x) = -log(p(x))

Where log() is the base-2 logarithm and p(x) is the probability of the event x.
The choice of the base-2 logarithm means that the units of the information measure is in bits (binary digits).

The calculation of information is often written as h(); for example:
h(x) = -log(p(x))

Low Probability Event: High Information (surprising).
High Probability Event: Low Information (unsurprising).
The negative sign ensures that the result is always positive or zero.

Information will be zero when the probability of an event is 1.0 or a certainty, e.g. there is no surprise.

Chain rule yields results

The chain rule provides the change in J w.r.t parameters w,b for each layers , Evaluation of $\frac{\partial J}{\partial w^{[L]}}$:

$$\frac{\partial J}{\partial a^{[L]}} = -\frac{1}{208} \times \sum_{k=1}^{208} \times \left[\frac{y}{a} - \frac{(1-y)}{(1-a)}\right]$$

$$\frac{\partial a}{\partial z^{[L]}} = \left[a(1-a)\right]$$

$$\frac{\partial z}{\partial w^{[L]}} = \frac{\partial\left[W^{\top}X+b\right]}{\partial w^{[L]}} = X^{\top}$$

$$\frac{\partial J}{\partial a^{[L]}} \times \frac{\partial a}{\partial z^{[L]}} \times \frac{\partial z}{\partial w^{[L]}} = \frac{\partial J}{\partial w^{[L]}}$$

$$\frac{\partial J}{\partial w^{[L]}} = -\frac{1}{208} \times \sum_{k=1}^{208} \times \left[\frac{y}{a} - \frac{(1-y)}{(1-a)}\right] \times \left[a(1-a)\right] \times \left[X^{\top}\right]$$

$$\frac{\partial J}{\partial w^{[L]}} = -\frac{1}{208} \times \sum_{k=1}^{208} \times \left[\frac{y-ya-a+ya}{a(1-a)}\right] \times \left[a(1-a)\right] \times \left[X^{\top}\right]$$

$$\frac{\partial J}{\partial w^{[L]}} = \frac{1}{208} \times \sum_{k=1}^{208} \times \left[a-y\right] \times \left[X^{\top}\right]$$

Check for dimensions :

Modeling inner product on given results :

$\left[a-y\right] \times \left[X^\top\right]$ has dimensions 1×12288 because,

$\left[a-y\right]$ has dimension :1×208.

$\left[X^\top\right]$ has dimension 208×12288.

$\left[(a-y)^\top\right]$ has dimension :208×1.

$\left[X\right]$ has dimension 12288×208.

$\left[X\right] \times \left[(a-y)^\top\right]$ has dimension 12288×1

What gradient looks like in this case

$$\nabla J(w_1, w_2, \ldots, w_{12288}) = \begin{bmatrix} \dfrac{\partial J}{\partial w_1}(w_1, w_2, \ldots, w_{12288}) \\ \dfrac{\partial J}{\partial w_2}(w_1, w_2, \ldots, w_{12288}) \\ \vdots \\ \dfrac{\partial J}{\partial w_{12288}}(w_1, w_2, \ldots, w_{12288}) \end{bmatrix}$$

We can see here dimension of gradient is 12288X1, just like $\left[X\right] \times \left[(a-y)^\top\right]$ has dimension 12288×1

What gradient looks like in this case

$$\frac{\partial J}{\partial W} = \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_{12288}} \end{bmatrix} = \begin{bmatrix} (a_1 - y_1) \times x_1 \\ (a_1 - y_1) \times x_2 \\ \vdots \\ (a_1 - y_1) \times x_{12288} \end{bmatrix}$$

We can see here dimension of gradient is 12288 X 1, just like $\left[X\right] \times \left[(a - y)^{\top}\right]$ has dimension 12288×1

Evaluating gradients :

The chain rule provides the change in J w.r.t parameters w,b for each layers , Evaluation of $\frac{\partial J}{\partial b^{[L]}}$:

$$\frac{\partial J}{\partial a^{[L]}} = -\frac{1}{208} \times \sum_{k=1}^{208} \times \left[\frac{y}{a} - \frac{(1-y)}{(1-a)}\right]$$

$$\frac{\partial a}{\partial z^{[L]}} = \left[a(1-a)\right]$$

$$\frac{\partial z}{\partial b^{[L]}} = \frac{\partial \left[W^\top X + b\right]}{\partial b^{[L]}} = 1$$

$$\frac{\partial J}{\partial a^{[L]}} \times \frac{\partial a}{\partial z^{[L]}} \times \frac{\partial z}{\partial b^{[L]}} = \frac{\partial J}{\partial b^{[L]}}$$

$$\frac{\partial J}{\partial b^{[L]}} = -\frac{1}{208} \times \sum_{k=1}^{208} \times \left[\frac{y}{a} - \frac{(1-y)}{(1-a)}\right] \times \left[a(1-a)\right] \times \left[1\right]$$

$$\frac{\partial J}{\partial b^{[L]}} = -\frac{1}{208} \times \sum_{k=1}^{208} \times \left[\frac{y-ya-a+ya}{a(1-a)}\right] \times \left[a(1-a)\right] \times \left[1\right]$$

$$\frac{\partial J}{\partial b^{[L]}} = \frac{1}{208} \times \sum_{k=1}^{208} \times \left[a-y\right] \times \left[1\right]$$

Chain rule yields results

The chain rule provides the change in J w.r.t parameters w,b for each layers :

$$\frac{\partial J}{\partial a^{[L]}} \times \frac{\partial a^{[L]}}{\partial z^{[L]}} \times \frac{\partial z^{[L]}}{\partial w^{[L]}} = \frac{\partial J}{\partial w^{[L]}}$$

$$\frac{\partial J}{\partial a^{[L]}} \times \frac{\partial a^{[L]}}{\partial z^{[L]}} \times \frac{\partial z^{[L]}}{\partial b^{[L]}} = \frac{\partial J}{\partial b^{[L]}}$$

$$\frac{\partial J}{\partial a^{[L]}} \times \frac{\partial a^{[L]}}{\partial z^{[L]}} \times \frac{\partial z^{[L]}}{\partial a^{[L-1]}} \times \frac{\partial a^{[L-1]}}{\partial z^{[L-1]}} \times \frac{\partial z^{[L-1]}}{\partial w^{[L-1]}} = \frac{\partial J}{\partial w^{[L-1]}}$$

$$\frac{\partial J}{\partial a^{[L]}} \times \frac{\partial a^{[L]}}{\partial z^{[L]}} \times \frac{\partial z^{[L]}}{\partial a^{[L-1]}} \times \frac{\partial a^{[L-1]}}{\partial z^{[L-1]}} \times \frac{\partial z^{[L-1]}}{\partial b^{[L-1]}} = \frac{\partial J}{\partial b^{[L-1]}}$$

$$\frac{\partial J}{\partial a^{[L]}} \times \frac{\partial a^{[L]}}{\partial z^{[L]}} \times \frac{\partial z^{[L]}}{\partial a^{[L-1]}} \times \frac{\partial a^{[L-1]}}{\partial z^{[L-1]}} \times \frac{\partial z^{[L-1]}}{\partial a^{[L-2]}} \times \frac{\partial a^{[L-2]}}{\partial z^{[L-2]}} \times \frac{\partial z^{[L-2]}}{\partial w^{[L-2]}} = \frac{\partial J}{\partial w^{[L-2]}}$$

$$\frac{\partial J}{\partial a^{[L]}} \times \frac{\partial a^{[L]}}{\partial z^{[L]}} \times \frac{\partial z^{[L]}}{\partial a^{[L-1]}} \times \frac{\partial a^{[L-1]}}{\partial z^{[L-1]}} \times \frac{\partial z^{[L-1]}}{\partial a^{[L-2]}} \times \frac{\partial a^{[L-2]}}{\partial z^{[L-2]}} \times \frac{\partial z^{[L-2]}}{\partial b^{[L-2]}} = \frac{\partial J}{\partial b^{[L-2]}}$$

Forward pass :

The chain rule provides the change in J w.r.t parameters w,b for each layers :

we store : $(\frac{\partial J}{\partial w^{[L]}}\ ,\ \frac{\partial J}{\partial b^{[L]}})$ as gradients.

Note :
$\frac{\partial J}{\partial b^{[L]}} = \frac{1}{208} \times \sum_{k=1}^{208} \times \left[a - y\right] \times \left[1\right]$

is a scalar value , summed over 208 examples.

What gradient looks like in this case

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{12288} \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{12288} \end{bmatrix} - \alpha \times \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_{12288}} \end{bmatrix}$$

We can see here dimension of gradient is 12288X1, just like $\left[X\right] \times \left[(a-y)^\top\right]$ has dimension 12288×1

What gradient looks like in this case

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{12288} \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{12288} \end{bmatrix} - \alpha \times \begin{bmatrix} (a_1 - y_1) \times x_1 \\ (a_2 - y_2) \times x_2 \\ \vdots \\ (a_{12288} - y_{12288}) \times x_{12288} \end{bmatrix}$$

We can see here dimension of gradient is 12288X1, just like $\left[X\right] \times \left[(a-y)^\top\right]$ has dimension 12288×1

Implementation

```python
def sigmoid(z):
    s = 1/(1+np.exp(-z))
    return s

def weight(dim):
    w = np.zeros((dim,1),dtype='float')
    b = 0.0
    return w,b

def propagate(X,w,b,Y) :
    m = train_set_x_flatten.shape[1]  # m=12288
    w,b = weight(m)
    A = sigmoid(np.dot(w.T,X)+b)  # A = [a1,a2,a3......a209]
    cost = (1/m)*np.sum(Y*np.log(A) + (1-Y)*np.log(1-A),keepdims=True)
    dw =(1/m)* np.matmul(X,(A-Y).T)
    db =(1/m)* np.sum((A-Y),axis=1)
    cost = np.squeeze(np.array(cost))
    grads = {"dw":dw,"db":db}
    return grads,cost
```

Implementation

```python
def optimize(X_train,Y_train,LEARNING_RATE=0.01,num_iterations=100,print_cost=False) :
    m =X_train.shape[1]
    w,b = weight(m)
    costs =[]
    for i in range(num_iterations) :
        grads,cost = propagate(X_train,w,b,Y_train)
        dw = grads["dw"]
        db = grads["db"]
        w[i] = w[i] -LEARNING_RATE*dw*X_train[i]
        b[i] = b[i] -LEARNING_RATE*db
        if i%100 == 0 :
            costs.append(cost)
        if print_cost :
            print(costs)
        params = {"w":w ,"b":b}
        grads = {"dw":dw,"db":db}
    return
```

Implementation

```
def predict (w,b,X):
    m = X.shape[1] # no of examples
    Y_prediction = np.zeros(1,m)
    w= w.rehsape(X.shape[1],1)
    A = sigmoid(np.dot(w.T,X)+b)
    for i in range(A.shape[1]) :
        if (A[0,i]) >0.5:
            Y_prediction[0,i] = 1
        else :
            Y_prediction[0,i] = 0
        return Y_prediction
```

Implementation :Model Development

```
def model(X_train, Y_train, X_test, Y_test, num_iterations=2000, learning_rate=0.5, print_co
    w, b = initialize_with_zeros(X_train.shape[0])
    params , grads , costs =optimize(w, b, X_train, Y_train, num_iterations, learning_rate, 
    w=params["w"]
    b=params["b"]
    Y_prediction_train = predict(w, b, X_train)
    Y_prediction_test = predict(w, b, X_test)

    # Print train/test Errors
    if print_cost:
        print("train accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_train - Y_trai
        print("test accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_test - Y_test))

    d = {"costs": costs,
         "Y_prediction_test": Y_prediction_test,
         "Y_prediction_train" : Y_prediction_train,
         "w" : w,
         "b" : b,
         "learning_rate" : learning_rate,
         "num_iterations": num_iterations}

    return d
```