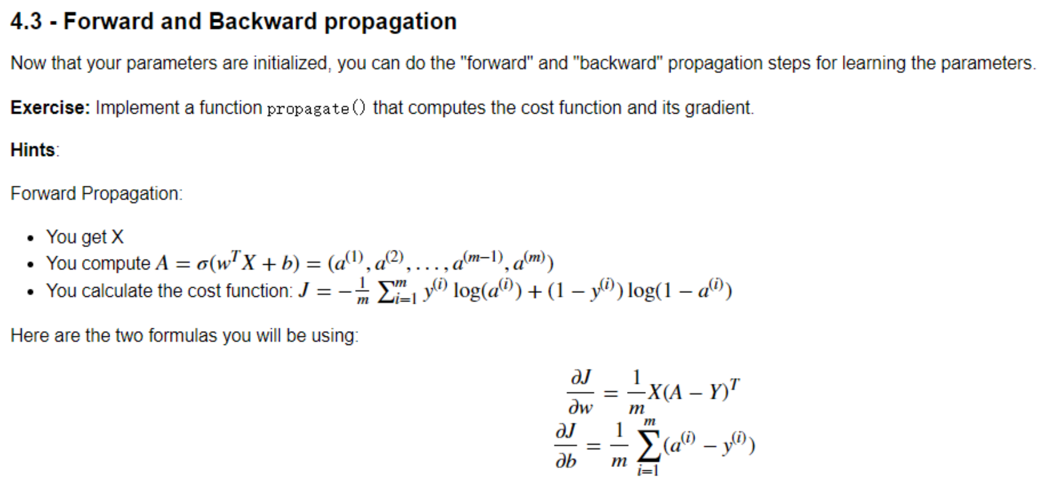
Week 2 - prog ex: NN for LR

笔记本:	DL 1 - NN and DL		
创建时间:	2021/1/7 02:16	**更新时间:**	2021/1/8 11:56



4.3 - Forward and Backward propagation

Now that your parameters are initialized, you can do the "forward" and "backward" propagation steps for learning the parameters.

Exercise: Implement a function `propagate()` that computes the cost function and its gradient.

Hints:

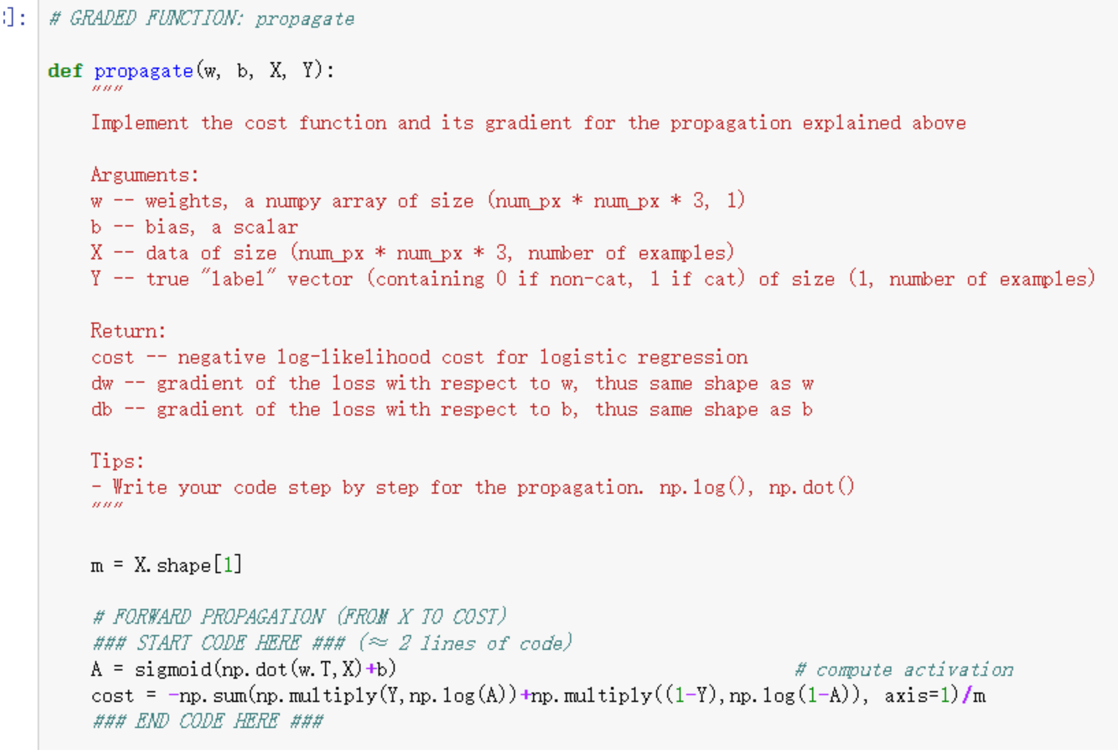
Forward Propagation:

- You get X
- You compute $A = \sigma(w^T X + b) = (a^{(1)}, a^{(2)}, \dots, a^{(m-1)}, a^{(m)})$
- You calculate the cost function: $J = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(a^{(i)}) + (1-y^{(i)})\log(1-a^{(i)})$

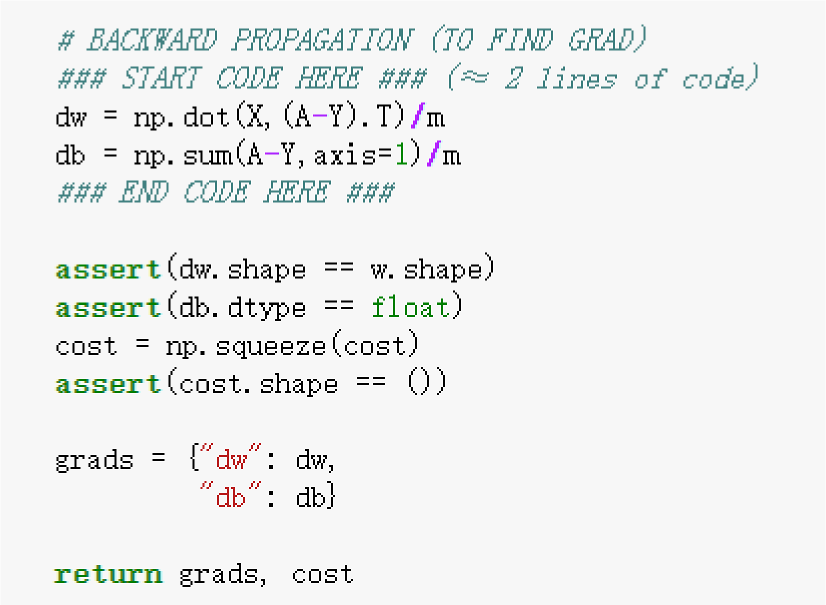
Here are the two formulas you will be using:

$$\frac{\partial J}{\partial w} = \frac{1}{m}X(A-Y)^T$$
$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)})$$

propagate

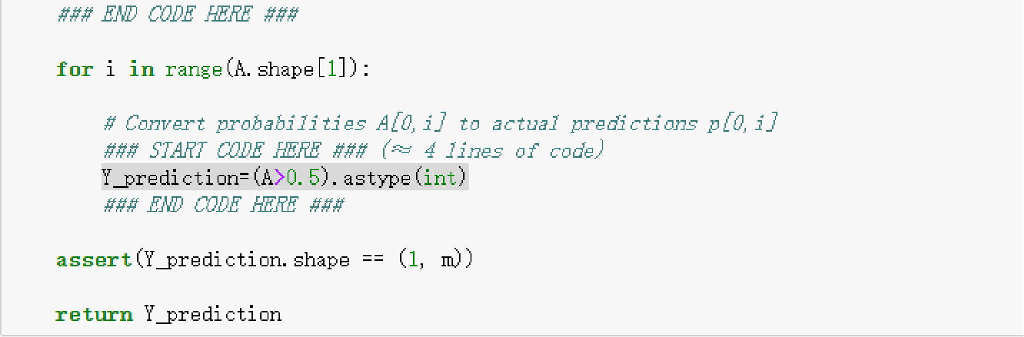


```
]: # GRADED FUNCTION: propagate

def propagate(w, b, X, Y):
    """
    Implement the cost function and its gradient for the propagation explained above

    Arguments:
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of size (num_px * num_px * 3, number of examples)
    Y -- true "label" vector (containing 0 if non-cat, 1 if cat) of size (1, number of examples)

    Return:
    cost -- negative log-likelihood cost for logistic regression
    dw -- gradient of the loss with respect to w, thus same shape as w
    db -- gradient of the loss with respect to b, thus same shape as b

    Tips:
    - Write your code step by step for the propagation. np.log(), np.dot()
    """

    m = X.shape[1]

    # FORWARD PROPAGATION (FROM X TO COST)
    ### START CODE HERE ### (≈ 2 lines of code)
    A = sigmoid(np.dot(w.T,X)+b)                                    # compute activation
    cost = -np.sum(np.multiply(Y,np.log(A))+np.multiply((1-Y),np.log(1-A)), axis=1)/m
    ### END CODE HERE ###
```



```
# BACKWARD PROPAGATION (TO FIND GRAD)
### START CODE HERE ### (≈ 2 lines of code)
dw = np.dot(X, (A-Y).T)/m
db = np.sum(A-Y,axis=1)/m
### END CODE HERE ###

assert(dw.shape == w.shape)
assert(db.dtype == float)
cost = np.squeeze(cost)
assert(cost.shape == ())

grads = {"dw": dw,
         "db": db}

return grads, cost
```

predict



```
### END CODE HERE ###

for i in range(A.shape[1]):

    # Convert probabilities A[0,i] to actual predictions p[0,i]
    ### START CODE HERE ### (≈ 4 lines of code)
    Y_prediction=(A>0.5).astype(int)
    ### END CODE HERE ###

assert(Y_prediction.shape == (1, m))

return Y_prediction
```