Logistic Regression with a neural network mindset

- to flatten an array X of (m, m, n) to $(m \times m \times n, 1)$

  `X_flatten = X.reshape(X.shape[0], -1).T`

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

    `s = 1/(1 + np.exp(-z))`

- `np.zeros([dim1, dim2])`
- **Forward propagation**
  - Activation $A = \sigma(w^T X + b) = (a^{(1)}, a^{(2)}, \ldots, a^{(m-1)}, a^{(m)})$
    - `sigmoid(np.dot(w.T, X) + b)`
  - Cost $J = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log(a^{(i)}) + (1-y^{(i)})\log(1-a^{(i)})$
    - `cost = -(np.dot(Y, np.log(A).T) + np.dot((1 - Y), (np.log(1 - A).T)))/m`
- **Backward propagation**
  - `dw = np.dot(X, (A - Y).T)/m`
  - `db = np.sum(A - Y)/m`
- `assert`
  - `assert(dw.shape == w.shape)`
  - `assert(db.dtype == float)`
  - `assert(cost.shape == ())`
  - `assert(isinstance(b, float) or isinstance(b, int))`

- Initialize (w,b)
- Optimize the loss iteratively to learn parameters (w,b):
  - computing the cost and its gradient
  - updating the parameters using gradient descent
- Use the learned (w,b) to predict the labels for a given set of examples
- Merge all functions into a model
  - ```
    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}
    ```
    ```