## 1. Packages

numpy, sklearn, matplotlib, testCases, planar_utils

## 2. Dataset

X, Y = load_planar_dataset()
shape_X = (2, 400) → 类似于坐标点 $(x_1, x_2)$
shape_Y = (1, 400) → 全部为0和1, 0为red, 1为blue
The dataset has 400 training examples.

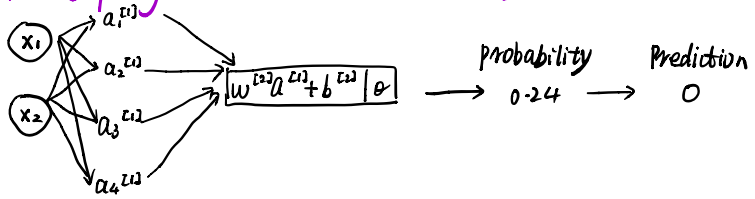
## 3. Simple Logistic Regression

clf = sklearn.linear_model.LogisticRegressionCV()
clf.fit(X.T, Y.T)

Accuracy ≈ 47%  △: Accuracy的计算式?

## 4. Neural Network model

### 4.1. Defining the neural network structure




```
def layer_sizes(X, Y):
    n_x = X.shape[0]   # input layer: 2
    n_h = 4            # hidden layer
    n_y = Y.shape[0]   # Output layer: 1
    return (n_x, n_h, n_y)
```

### 4.2. Initialize the model's parameters

```
def initialize_parameters(n_x, n_h, n_y):
    W1 = np.random.randn(n_h, n_x) * 0.01   # (4, 2)
    b1 = np.zeros((n_h, 1))                 # (4, 1)
    W2 = np.random.randn(n_y, n_h) * 0.01   # (1, 4)
    b2 = np.zeros((n_y, 1))                 # (1, 1)
    return parameters.
```

### 4.3. Loop

#### 4.3.1 Forward propagation

```
def forward_propagation(X, parameters):
    W1 = parameters["W1"]
    W2 ..... "W2"
    b1 ..... "b1"
    b2 ..... "b2"
    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)
    return A2, cache
```

$$
\begin{cases}
Z^{[1](i)} = W^{[1]}X^{(i)} + b^{[1]} \\
a^{[1](i)} = \tanh(z^{[1](i)}) \\
z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]} \\
\hat{y}^{(i)} = a^{[2](i)} = \theta(z^{[2](i)}) \\
\hat{y}^{(i)}_{pred} = \begin{cases} 1 & \text{if } a^{[2](i)} > 0.5 \\ 0 & \text{otherwise} \end{cases}
\end{cases}
$$

→ a dictionary containing "Z1", "A1", "Z2" and "A2".

The sigmoid output of the second activation

### 4.3.2. Cost

cross-entropy loss: $J = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right)$

($A^{[2]}$ loss)

```
def compute_loss(A2, Y, parameters):
    m = Y.shape[1]  # number of examples
    logprobs = np.dot(np.log(A2), Y.T) + np.dot(np.log(1-A2), (1-Y).T)
    cost = (-1/m) * logprobs
    cost = float(np.squeeze(cost))
    return cost.
```

### 4.3.3. backward propagation

```
def backward_propagation(parameters, cache, X, Y)
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = ..... ["W2"]
    A1 = cache["A1"]
    A2 = cache["A2"]
    dZ2 = A2 - Y
    dW2 = 1/m * np.dot(dZ2, A1.T)
    db2 = np.sum(dZ2, axis=1, keepdims=True)
    dZ1 = np.dot(W2.T, dZ2) * (1 - power(A1,2))
    dW1 = 1/m * np.dot(dZ1, X.T)
    db1 = np.sum(dZ1, axis=1, keepdims=True)
    return grads.
```

$$\begin{cases} dZ^{[2]} = A^{[2]} - Y \\ dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T} \\ db^{[2]} = np.sum(dZ^{[2]}, axis=1, ...) \\ dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]}) \\ dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T \\ db^{[1]} = np.sum(dZ^{[1]}, ...) \end{cases}$$

### 4.3.4. update parameters

```
def update_parameters(parameters, grads, learning_rate=1.2):
    W1 = parameters["W1"]
    W2 = ..... ["W2"]
    b1 = ..... ["b1"]
    b2 = ..... ["b2"]
    dW1 = grads["dW1"]
    db1 = ..... ["db1"]
    dW2 = ..... ["dW2"]
    db2 = ..... ["db2"]
    W1 = W1 - learning_rate * dW1
```

```
    W2 = W2   ...
    b1 = b1   ...
    b2 = b2   ...
    return parameters
```

4.4. Integrate parts 4.1, 4.2, 4.3 in nn_model()

```
def nn_model(X, Y, n_h, num_iterations = 10000, print_cost=False):
    np.random.seed(3)
    n_x = layer_sizes(X, Y)[0]
    n_y = layer_sizes(X, Y)[2]
    # Initialize parameters
    parameters = initialize_parameters(n_x, n_h, n_y)
    # Loop
    for i in range(0, num_iterations):
    # Forward propagation.
    A2, cache = forward_propagation(X, parameters)
    # Cost function
    cost = compute_cost(A2, Y, parameters)
    # Backward propagation
    grads = backward_propagation(parameters, cache, X, Y)
    # Gradient descent update
    parameters = update_parameters(parameters, grads)

    return parameters
```

4.5. Predictions

```
def predict(parameters, X):
    A2, cache = forward_propagation(X, parameters)
    predictions = (A2 > 0.5)

    return predictions
```

$$\text{predictions} = y_{\text{prediction}} = \mathbb{1}\{\text{activation} > 0.5\} = \begin{cases} 1 & \text{if } > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

forward_propagation(X, parameters)

↓

使用 nn_model 训练出来的 parameters.

这里 Neural Network 的 Accuracy 为 90%.