Planar data classification with one hidden layer

- use *sklearn's* built-in functions to do logistic linear regression
  - ```
    clf = sklearn.linear_model.LogisticRegressionCV();
    clf.fit(X.T, Y.T);
    LR_predictions = clf.predict(X.T)
    ```

- `np.tanh()`
- `np.squeeze()`
  - ```
    cost = -((np.dot(Y, np.log(A2).T) + np.dot(1 - Y, np.log(1 - A2).T))/m)
    type(cost)
    print(cost)
    ```

    ```
    <class 'numpy.ndarray'>
    [[0.69291989]]
    ```

  - ```
    cost = -((np.dot(Y, np.log(A2).T) + np.dot(1 - Y, np.log(1 - A2).T))/m)
    cost = np.squeeze(cost)
    type(cost)
    print(cost)
    ```

    ```
    <class 'numpy.ndarray'>
    0.6929198937761264
    ```

- `assert(isinstance(cost, float))`
  - ```
    cost = -((np.dot(Y, np.log(A2).T) + np.dot(1 - Y, np.log(1 - A2).T))/m)
    cost = np.squeeze(cost)
    print(cost.dtype)
    assert(isinstance(cost, float))
    ```

    ```
    float64
    AssertionError
    ```

  - ```
    logprobs = np.multiply(np.log(A2), Y) + np.multiply((1 - Y),
                           np.log(1 - A2))
    cost = - np.sum(logprobs) / m
    print(type(cost))
    print(cost.dtype)
    assert(isinstance(cost, float))
    ```

    ```
    <class 'numpy.float64'>
    float64
    0.6929198937761266
    ```
    ```

    - ```
      cost = -((np.dot(Y, np.log(A2).T) + np.dot(1 - Y, np.log(1 - A2).T))/m)
      cost = np.squeeze(cost)
      cost = np.float64(cost)
      print(type(cost))
      print(cost.dtype)
      assert(isinstance(cost, float))
      ```

      ```
      <class 'numpy.float64'>
      float64
      0.6929198937761264
      ```

- $g^{[1]'}(Z^{[1]})$
    - since $a = g^{[1]}(z) = tanh(z)$
    - $g^{[1]'}(z) = 1 - a^2$
    - `(1 - np.power(A1, 2))`
- predictions
    - ```
      A2, cache = forward_propagation(X, parameters)
      predictions = np.round(A2)
      ```