Problem Statement: You have just been hired as an AI expert by the French Football Corporation. They would like you to recommend positions where France's goal keeper should kick the ball so that the French team's players can then hit it with their head.

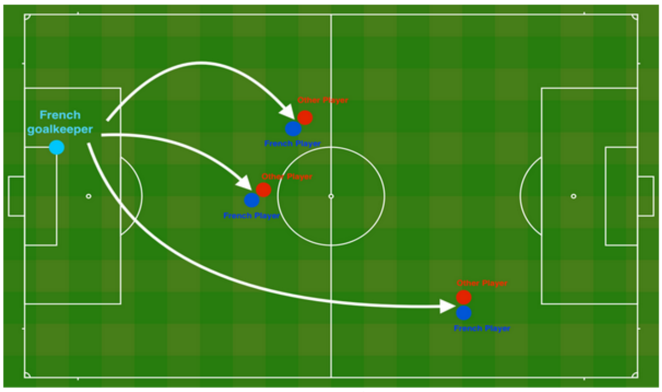
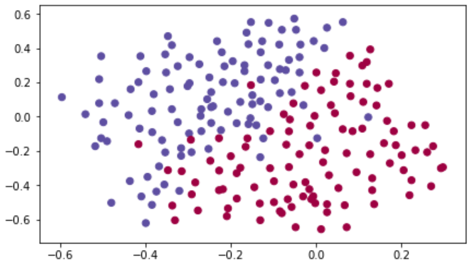


Figure 1 : **Football field**
The goal keeper kicks the ball in the air, the players of each team are fighting to hit the ball with their head

They give you the following 2D dataset from France's past 10 games.

```
train_X, train_Y, test_X, test_Y = load_2D_dataset()
```



Each dot corresponds to a position on the football field where a football player has hit the ball with his/her head after the French goal keeper has shot the ball from the left side of the football field.

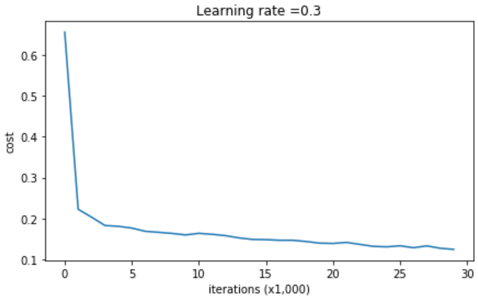
- If the dot is blue, it means the French player managed to hit the ball with his/her head
- If the dot is red, it means the other team's player hit the ball with their head

Your goal: Use a deep learning model to find the positions on the field where the goalkeeper should kick the ball.

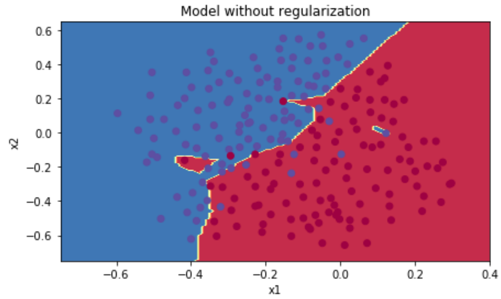
Let's train the model without any regularization, and observe the accuracy on the train/test sets.

```
parameters = model(train_X, train_Y)
print ("On the training set:")
predictions_train = predict(train_X, train_Y, parameters)
print ("On the test set:")
predictions_test = predict(test_X, test_Y, parameters)
```

```
Cost after iteration 0: 0.6557412523481002
Cost after iteration 10000: 0.16329987525724216
Cost after iteration 20000: 0.13851642423255986
```




```
On the training set:
Accuracy: 0.947867298578
On the test set:
Accuracy: 0.915
```



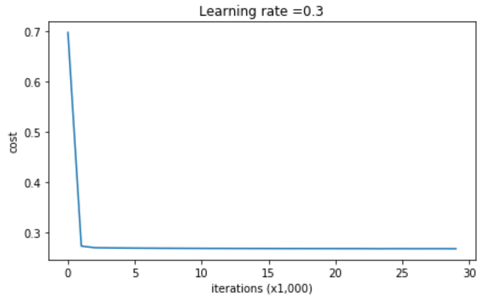
The non-regularized model is obviously overfitting the training set. It is fitting the noisy points! Lets now look at two techniques to reduce overfitting.

Let's now run the model with L2 regularization $(\lambda = 0.7)$. The `model()` function will call:

- `compute_cost_with_regularization` instead of `compute_cost`
- `backward_propagation_with_regularization` instead of `backward_propagation`

```
parameters = model(train_X, train_Y, lambd = 0.7)
print ("On the train set:")
predictions_train = predict(train_X, train_Y, parameters)
print ("On the test set:")
predictions_test = predict(test_X, test_Y, parameters)
```

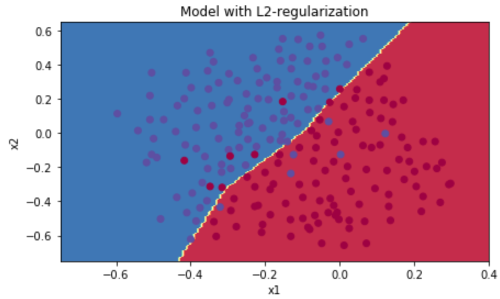
```
Cost after iteration 0: 0.6974484493131264
Cost after iteration 10000: 0.2684918873282239
Cost after iteration 20000: 0.2680916337127301
```




```
On the train set:
Accuracy: 0.938388625592
On the test set:
Accuracy: 0.93
```

Congrats, the test set accuracy increased to 93%. You have saved the French football team!

You are not overfitting the training data anymore. Let's plot the decision boundary.



Observations:

- The value of λ is a hyperparameter that you can tune using a dev set.
- L2 regularization makes your decision boundary smoother. If λ is too large, it is also possible to "oversmooth", resulting in a model with high bias.

What is L2-regularization actually doing?:

L2-regularization relies on the assumption that a model with small weights is simpler than a model with large weights. Thus, by penalizing the square values of the weights in the cost function you drive all the weights to smaller values. It becomes too costly for the cost to have large weights! This leads to a smoother model in which the output changes more slowly as the input changes.