In [ ]:
```
## Deep Learning Course (980)
## Assignment One

__Assignment Goals__:
- Start with TensorFlow (version 1.0).
- Implement and apply a multi-layer feed-forward neural network classifie
- Understand the differences and trade-offs between linear regression, lo

In this assignment, you will be asked to install TensorFlow and Jupyter N

__DataSet__: dataset has 100 instances and two features.

1. Install TensorFlow (1.15.0) and Jupyter Notebook. (15 points)
Run the provided code [Linear Regression](#linear_regression). This code
2. Using code similar to what was provided for linear regression, impleme
Hint: What is the correct loss function for logistic regression compared
3. Implement a multi-layer feed-forward neural net and try to reach 100 a
4. Use tf.keras to implement the exact graph that you Implemented in the

__Submission Notes__:

Please use Jupyter Notebook. The notebook should include the final code,

You can use visualize() helper function to visualize the model's decision

__Instructions__:

The university policy on academic dishonesty and plagiarism (cheating) wi

Your assignments will be marked based on correctness, originality (the im
```

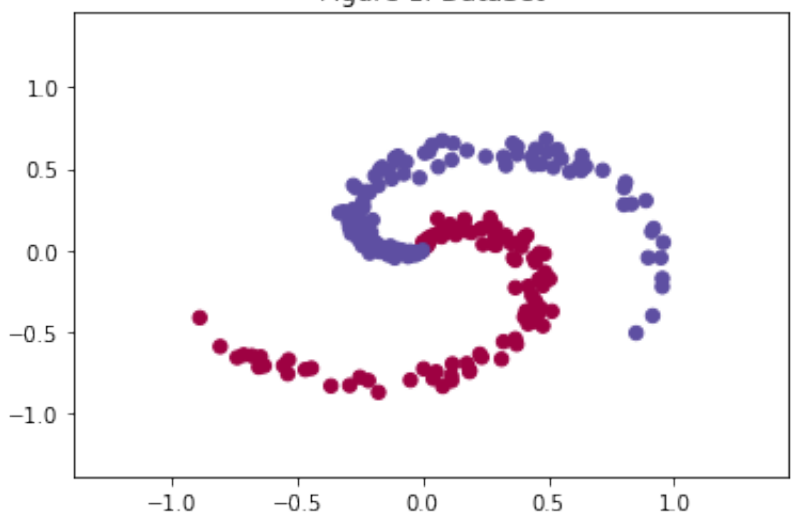
In [7]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
%matplotlib inline
#this line makes the notebook put the figures in-line rather than genera
```

In [8]:

```
# helper function for geterating the data
def data_generator(N=100, D=2, K=2):
    # N number of points per class; D dimensionality; k number of classe
    np.random.seed(0)
    X = np.zeros((N * K, D))
    y = np.zeros((N * K), dtype='uint8')
    for j in range(K):
        ix = range(N * j, N * (j + 1))
        r = np.linspace(0.0, 1, N)  # radius
        t = np.linspace(j * 4, (j + 1) * 4, N) + np.random.randn(N) * 0.
        X[ix] = np.c_[r * np.sin(t), r * np.cos(t)]
        y[ix] = j
    fig = plt.figure()
    plt.title('Figure 1: DataSet')
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)

    plt.xlim(X.min() - .5, X.max() + .5)
    plt.ylim(X.min() - .5, X.max() + .5)
    return X, y


# helper function for visualizing the boundaries
def visualize(sample, target, predict, se):
    """
    function for visualizing the classifier boundaries on the TOY datase

    @param sample: Training data features
    @param target: Target
    @param predict: Model prediction
    @param se: The model's session
    """

    h = 0.02
    x_min, x_max = sample[:, 0].min() - 1, sample[:, 0].max() + 1
    y_min, y_max = sample[:, 1].min() - 1, sample[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = np.round(se.run(predict, {X: (np.c_[xx.ravel(), yy.ravel()])}))
    Z = Z.reshape(xx.shape)
    fig = plt.figure()
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)
    plt.scatter(sample[:, 0], sample[:, 1], c=target, s=40, cmap=plt.cm.
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
```

In [9]:
```
#  TOY DataSet
sample, target = data_generator()
# print(target.shape)
```

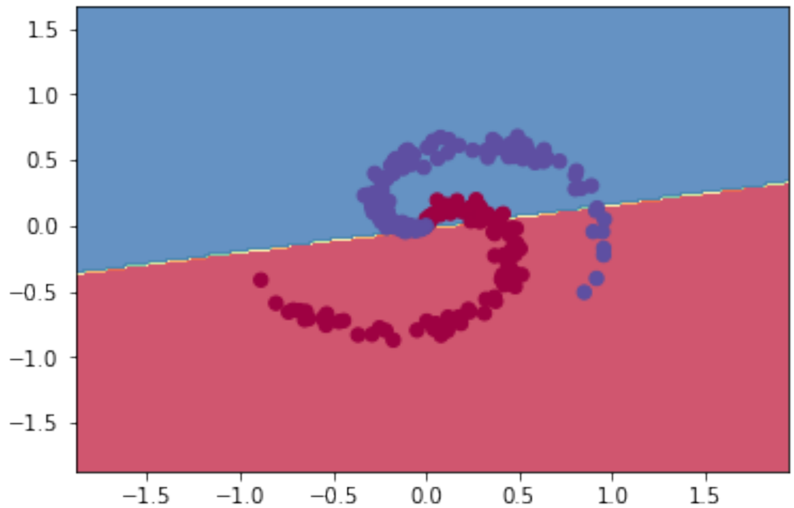
Figure 1: DataSet



In [10]:
```
tf.set_random_seed(1)

# Almost-correct Linear Regression
X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W = tf.Variable(tf.random_normal(shape=[2, 1], seed=1))
b = tf.Variable(tf.random_uniform([1]), name="bias")

m = X.shape[0]

first_layer = (tf.matmul(X, W)) + b  ## Y predicted
objective_function = tf.reduce_mean((tf.square(first_layer-Y)))

# the reduce_sum seems to operate on just 1 element

LR = tf.train.GradientDescentOptimizer(learning_rate=.5).minimize(object

# predicted value above 0.5 -> predict = 1 = classify as positive
predict = tf.cast(tf.greater(first_layer, 0.5), tf.float32)

accu = tf.reduce_mean(tf.cast(tf.equal(predict, Y), tf.float32))

with tf.Session() as se:
    se.run(tf.global_variables_initializer())
    for i in range(100):
        se.run(LR,{X : sample,Y: target.reshape(-1,1)})
        # -1 is like "unspecified"
```

```
        # -1 is like "unspecified"
        print("Epoch:", (i + 1),"loss:", "{:.3f}".format(se.run(objectiv
              "acc:", "{:.3f}".format(se.run(accu,{X:sample,Y:target.res
    visualize(sample, target, predict, se)
```

```
Epoch: 97 loss: 0.140 acc: 0.750
Epoch: 98 loss: 0.140 acc: 0.750
Epoch: 99 loss: 0.140 acc: 0.750
Epoch: 100 loss: 0.140 acc: 0.750
```



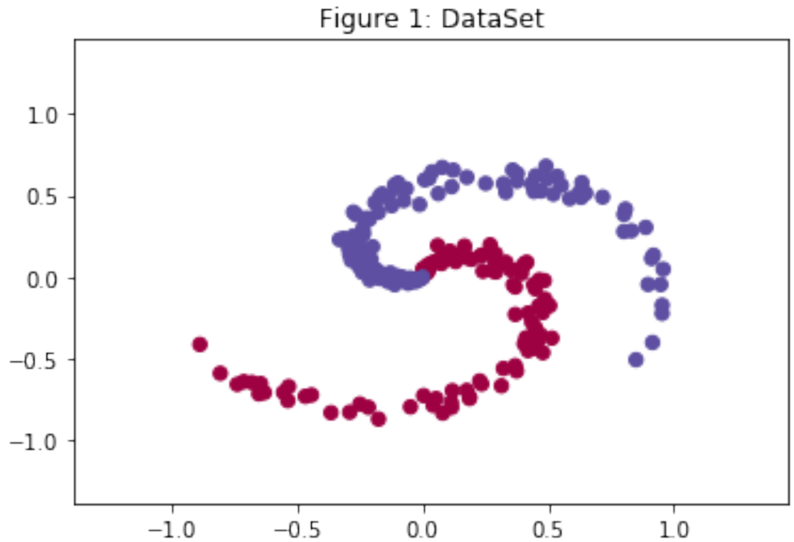
In [ ]:

In [38]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
%matplotlib inline
#this line makes the notebook put the figures in-line rather than genera
```

In [39]:

```
# helper function for geterating the data
def data_generator(N=100, D=2, K=2):
    # N number of points per class; D dimensionality; k number of classe
    np.random.seed(0)
    X = np.zeros((N * K, D))
    y = np.zeros((N * K), dtype='uint8')
    for j in range(K):
        ix = range(N * j, N * (j + 1))
        r = np.linspace(0.0, 1, N)  # radius
        t = np.linspace(j * 4, (j + 1) * 4, N) + np.random.randn(N) * 0.
        X[ix] = np.c_[r * np.sin(t), r * np.cos(t)]
        y[ix] = j
    fig = plt.figure()
    plt.title('Figure 1: DataSet')
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)

    plt.xlim(X.min() - .5, X.max() + .5)
    plt.ylim(X.min() - .5, X.max() + .5)
    return X, y


# helper function for visualizing the boundaries
def visualize(sample, target, predict, se):
    """
    function for visualizing the classifier boundaries on the TOY datase

    @param sample: Training data features
    @param target: Target
    @param predict: Model prediction
    @param se: The model's session
    """

    h = 0.02
    x_min, x_max = sample[:, 0].min() - 1, sample[:, 0].max() + 1
    y_min, y_max = sample[:, 1].min() - 1, sample[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = np.round(se.run(predict, {X: (np.c_[xx.ravel(), yy.ravel()])}))
    Z = Z.reshape(xx.shape)
    fig = plt.figure()
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)
    plt.scatter(sample[:, 0], sample[:, 1], c=target, s=40, cmap=plt.cm.
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
```

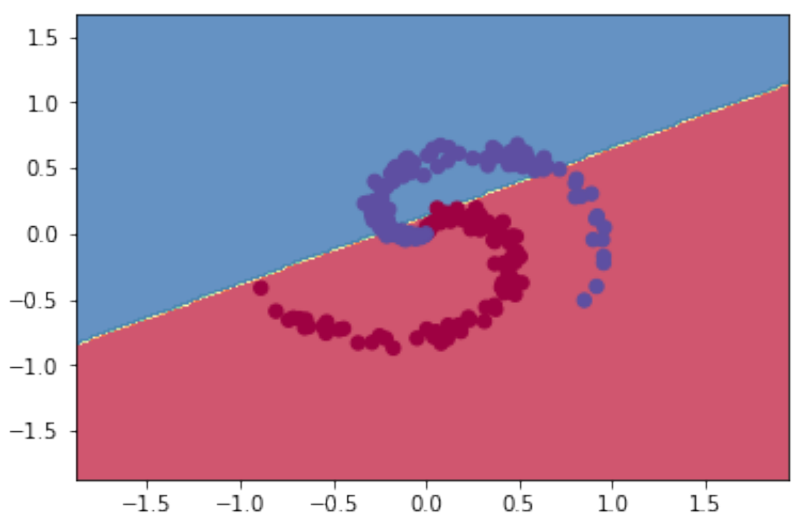
In [40]:
```
sample, target = data_generator()
```



In [42]:
```
tf.set_random_seed(1)

# Almost-correct Linear Regression
X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal(shape=[2, 1], seed=1))
b = tf.Variable(tf.random_uniform([1]), name="bias")


first_layer = tf.nn.sigmoid(tf.matmul(X, W) + b)
objective_function = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_log

# the reduce_sum seems to operate on just 1 element

LR = tf.train.GradientDescentOptimizer(learning_rate=.9).minimize(object

#predictions = tf.equal(tf.argmax(first_layer, 1), tf.argmax(Y, 1))
#accuracy = tf.reduce_mean(tf.cast(predictions, tf.float32))

delta = tf.abs((Y - first_layer))
predict = tf.cast(tf.greater(first_layer, tf.constant(0.5)), tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predict,Y), tf.float32))

# predicted value above 0.5 -> predict = 1 = classify as positive

epochs = 10000
with tf.Session() as se:
    se.run(tf.global_variables_initializer())
    for i in range(epochs):
        dict_val = {X: sample, Y: target.reshape(-1, 1)}
        _ = se.run(LR, feed_dict={X: sample, Y: target.reshape(-1, 1)})
```

```
        if i % 500 == 0:
            loss = objective_function.eval(dict_val)
            acc = accuracy.eval(dict_val)
            print("Epoch:", (i), "loss:", "{:.3f}".format(loss), "accura
    visualize(sample, target, predict, se)
```

```
Epoch: 0 loss: 0.663 accuracy = 0.680
Epoch: 500 loss: 0.613 accuracy = 0.795
Epoch: 1000 loss: 0.604 accuracy = 0.795
Epoch: 1500 loss: 0.599 accuracy = 0.805
Epoch: 2000 loss: 0.596 accuracy = 0.805
Epoch: 2500 loss: 0.594 accuracy = 0.805
Epoch: 3000 loss: 0.592 accuracy = 0.810
Epoch: 3500 loss: 0.591 accuracy = 0.810
Epoch: 4000 loss: 0.590 accuracy = 0.810
Epoch: 4500 loss: 0.589 accuracy = 0.815
Epoch: 5000 loss: 0.588 accuracy = 0.810
Epoch: 5500 loss: 0.588 accuracy = 0.810
Epoch: 6000 loss: 0.587 accuracy = 0.810
Epoch: 6500 loss: 0.587 accuracy = 0.810
Epoch: 7000 loss: 0.586 accuracy = 0.815
Epoch: 7500 loss: 0.586 accuracy = 0.815
Epoch: 8000 loss: 0.585 accuracy = 0.815
Epoch: 8500 loss: 0.585 accuracy = 0.815
Epoch: 9000 loss: 0.585 accuracy = 0.815
Epoch: 9500 loss: 0.584 accuracy = 0.815
```



In [ ]:

In [ ]:

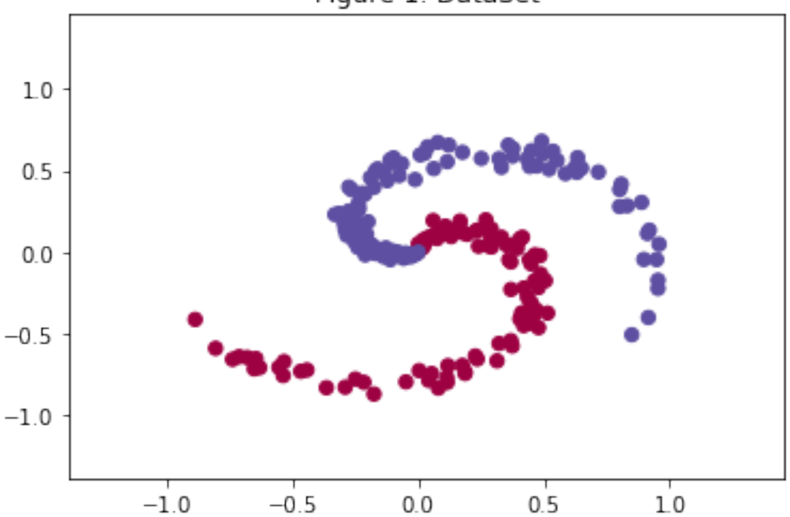
In [20]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
%matplotlib inline

#this line makes the notebook put the figures in-line rather than genera
```

In [21]:
```

# helper functions
# helper function for geterating the data
def data_generator(N=100, D=2, K=2):
    # N number of points per class; D dimensionality; k number of classe
    np.random.seed(0)
    X = np.zeros((N * K, D))
    y = np.zeros((N * K), dtype='uint8')
    for j in range(K):
        ix = range(N * j, N * (j + 1))
        r = np.linspace(0.0, 1, N)  # radius
        t = np.linspace(j * 4, (j + 1) * 4, N) + np.random.randn(N) * 0.
        X[ix] = np.c_[r * np.sin(t), r * np.cos(t)]
        y[ix] = j
    fig = plt.figure()
    plt.title('Figure 1: DataSet')
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)

    plt.xlim(X.min() - .5, X.max() + .5)
    plt.ylim(X.min() - .5, X.max() + .5)
    return X, y


# helper function for visualizing the boundaries
# helper function for visualizing the boundaries
def visualize(sample, target, model, se):
    """
    function for visualizing the classifier boundaries on the TOY datase

    @param sample: Training data features
    @param target: Target
    @param predict: Model prediction
    @param se: The model's session
    """

    h = 0.02
    x_min, x_max = sample[:, 0].min() - 1, sample[:, 0].max() + 1
    y_min, y_max = sample[:, 1].min() - 1, sample[:, 1].max() + 1
```

```
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    XX = {X: (np.c_[xx.ravel(), yy.ravel()])}
    Z = np.round(model.predict(np.c_[xx.ravel(), yy.ravel()], steps=64))
    Z = Z.reshape(xx.shape)
    fig = plt.figure()
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)
    plt.scatter(sample[:, 0], sample[:, 1], c=target, s=40, cmap=plt.cm.
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
```

In [22]:
```
sample, target = data_generator()
```

Figure 1: DataSet



In [23]:
```
tf.set_random_seed(1)

# Almost-correct Linear Regression
X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal(shape=[2, 1], seed=1))
b = tf.Variable(tf.random_uniform([1]), name="bias")

xx = sample
yy = target.reshape(-1, 1)


def run_model():
    # define model
    model = Sequential()
    model.add(Dense(16, activation='relu', kernel_initializer='he_normal
    model.add(Dense(12, activation='relu', kernel_initializer='he_normal
    model.add(Dense(4, activation='relu', kernel_initializer='he_normal'
    model.add(Dense(1, activation='sigmoid'))
    # compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
    return model


epochs = 10
with tf.Session() as se:
    se.run(tf.global_variables_initializer())
    for i in range(epochs):
        model = run_model()
        # fit the model
        model.fit(xx, yy, epochs=epochs, steps_per_epoch=100, batch_size
        # evaluate the model
        loss, acc = model.evaluate(xx, yy, verbose=0)
        # print('Test Accuracy: %.8f' % acc)
        print("Epoch:", (i), "loss:", "{:.3f}".format(loss), "accuracy =

    #visualize(sample, target, model, se)
```

```
Epoch: 0 loss: 0.017 accuracy = 0.995000004768372
Epoch: 1 loss: 0.026 accuracy = 0.995000004768372
Epoch: 2 loss: 0.019 accuracy = 0.995000004768372
Epoch: 3 loss: 0.020 accuracy = 0.995000004768372
Epoch: 4 loss: 0.141 accuracy = 0.995000004768372
Epoch: 5 loss: 0.021 accuracy = 0.995000004768372
Epoch: 6 loss: 0.151 accuracy = 0.995000004768372
Epoch: 7 loss: 0.016 accuracy = 0.995000004768372
Epoch: 8 loss: 0.165 accuracy = 0.995000004768372
Epoch: 9 loss: 0.024 accuracy = 0.995000004768372
```

In [ ]:

In [ ]:

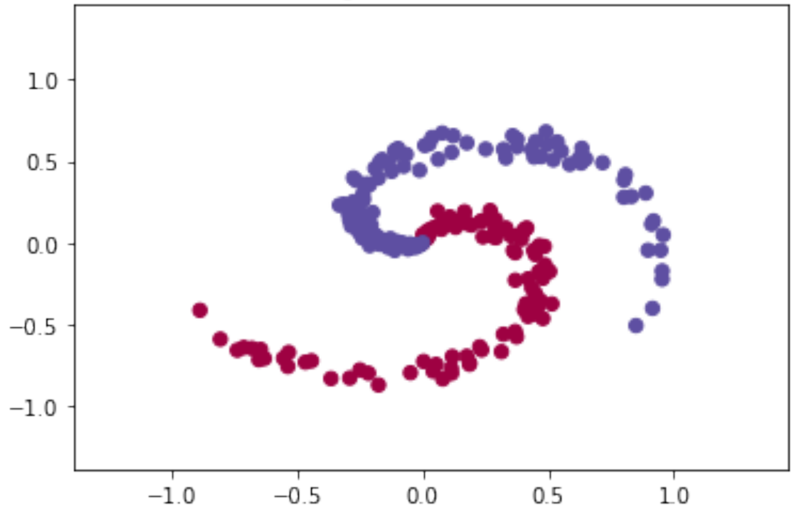
In [20]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
%matplotlib inline
#this line makes the notebook put the figures in-line rather than genera
```

In [21]:

```
# helper function for geterating the data
def data_generator(N=100, D=2, K=2):
    # N number of points per class; D dimensionality; k number of classe
    np.random.seed(0)
    X = np.zeros((N * K, D))
    y = np.zeros((N * K), dtype='uint8')
    for j in range(K):
        ix = range(N * j, N * (j + 1))
        r = np.linspace(0.0, 1, N)  # radius
        t = np.linspace(j * 4, (j + 1) * 4, N) + np.random.randn(N) * 0.
        X[ix] = np.c_[r * np.sin(t), r * np.cos(t)]
        y[ix] = j
    fig = plt.figure()
    plt.title('Figure 1: DataSet')
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)

    plt.xlim(X.min() - .5, X.max() + .5)
    plt.ylim(X.min() - .5, X.max() + .5)
    return X, y


# helper function for visualizing the boundaries
def visualize(sample, target, predict, se):
    """
    function for visualizing the classifier boundaries on the TOY datase

    @param sample: Training data features
    @param target: Target
    @param predict: Model prediction
    @param se: The model's session
    """

    h = 0.02
    x_min, x_max = sample[:, 0].min() - 1, sample[:, 0].max() + 1
    y_min, y_max = sample[:, 1].min() - 1, sample[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = np.round(se.run(predict, {X: (np.c_[xx.ravel(), yy.ravel()])}))
    Z = Z.reshape(xx.shape)
    fig = plt.figure()
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)
    plt.scatter(sample[:, 0], sample[:, 1], c=target, s=40, cmap=plt.cm.
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
```

In [22]:
```
sample, target = data_generator()
```

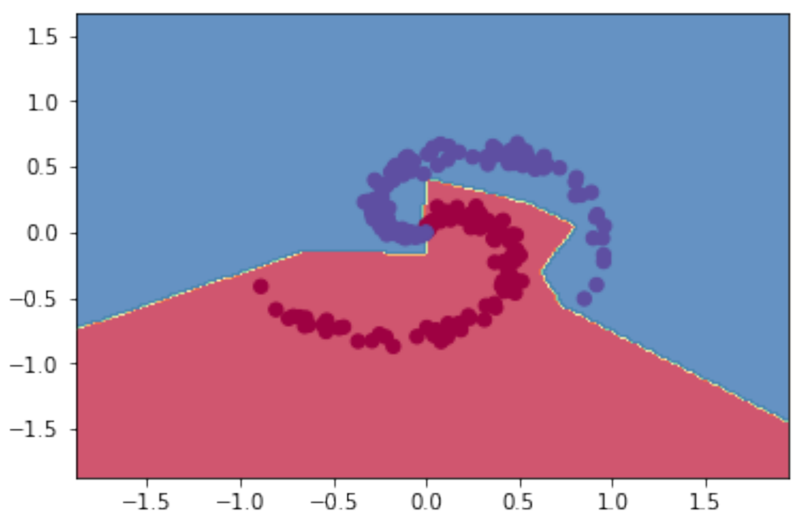
Figure 1: DataSet



In [23]:
```
# tf.set_random_seed(1)

# Almost-correct Linear Regression
X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal(shape=[2, 1], seed=1))
b = tf.Variable(tf.random_uniform([1]), name="b")

def sigmoid(x):
    return 1/(1+tf.math.exp(-x))


n_hidden1 = 5
n_hidden2 = 5
n_input = 2


# Weight and Biases first hidden layer
w1 = tf.Variable(tf.random_normal([n_input, n_hidden1], seed=1))
b1 = tf.Variable(tf.random_uniform([n_hidden1]))

# Weight and Biases second hidden layer
w2 = tf.Variable(tf.random_normal([n_hidden1, n_hidden2], seed=1))
b2 = tf.Variable(tf.random_uniform([n_hidden2]))

# Weight and Biases second hidden layer
w3 = tf.Variable(tf.random_normal([n_hidden2, 1], seed=1))
b3 = tf.Variable(tf.random_uniform([1]))


layer_1 = tf.add(tf.matmul(X, w1), b1)
```

```
layer_1 = tf.maximum(0.0, layer_1)
layer_2 = tf.add(tf.matmul(layer_1, w2), b2)
layer_2 = tf.maximum(0.0, layer_2)
out = tf.add(tf.matmul(layer_2, w3), b3)

loss = tf.reduce_mean(-Y*tf.math.log(sigmoid(out)) - (1-Y)* tf.math.log(
LR = tf.train.AdamOptimizer(learning_rate=0.005).minimize(loss)

predict = tf.cast(tf.greater(out, 0.5), tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predict,Y), tf.float32))

epochs = 2000

se = tf.Session()
se.run(tf.global_variables_initializer())

for i in range(epochs):
    se.run(LR,{X : sample,Y: target.reshape(-1,1)})
    print("Epoch:", (i + 1),"loss:", "{:.3f}".format(se.run(loss,{X:samp
visualize(sample,target, predict,se)
```

```
Epoch: 1997 loss: 0.013 acc: 0.995
Epoch: 1998 loss: 0.013 acc: 0.995
Epoch: 1999 loss: 0.013 acc: 0.995
Epoch: 2000 loss: 0.013 acc: 0.995
```



In [ ]:

In [ ]: