In [7]:
```
#Form a 3 layer neural network (one input, one hidden, and one output) to learn the XOR function.
#The input layer should contain 2 binary inputs.
#Second Layer (first hidden layer) should contain 3 neurons.
#Output layer contains 1 neuron which will produce the output of the XOR function
```

In [8]:
```
import numpy as np

# Activation Functions
def tanh(x):
    return np.tanh(x)

def d_tanh(x):
    return 1 - np.square(np.tanh(x))

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

def d_sigmoid(x):
    return (1 - sigmoid(x)) * sigmoid(x)

# Loss Functions
def logloss(y, a):
    return -(y*np.log(a) + (1-y)*np.log(1-a))

def d_logloss(y, a):
    return (a - y)/(a*(1 - a))
```

In [9]:
```
# The Layer class
class Layer:

    activationFunctions = {
        'tanh': (tanh, d_tanh),
        'sigmoid': (sigmoid, d_sigmoid)
    }
    learning_rate = 0.1

    def __init__(self, inputs, neurons, activation):# inputs to a layer=the number of neurons in the previous layer
                                                    # neurons- the number of neurons in a layer
                                                    # activation - the particular activation for a layer
        self.W = np.random.randn(neurons, inputs)
        self.b = np.zeros((neurons, 1))
        self.act, self.d_act = self.activationFunctions.get(activation)

    def feedforward(self, A_prev):
        self.A_prev = A_prev
        self.Z = np.dot(self.W, self.A_prev) + self.b
        self.A = self.act(self.Z)
        return self.A

    def backprop(self, dA):
        dZ = np.multiply(self.d_act(self.Z), dA)
        dA_prev = np.dot(self.W.T, dZ)

        dW = 1/dZ.shape[1] * np.dot(dZ, self.A_prev.T)
        db = 1/dZ.shape[1] * np.sum(dZ, axis=1, keepdims=True)

        self.W = self.W - self.learning_rate * dW
        self.b = self.b - self.learning_rate * db

        return dA_prev
```

In [10]:
```
x_train = np.array([[0, 0, 1, 1], [0, 1, 0, 1]]) # dim x m
y_train = np.array([[0, 1, 1, 0]]) # 1 x m

#dnn = DeepNeuralNetwork(sizes=[784, 128, 64, 10])

m = 4
epochs = 3000

layers = [Layer(2, 3, 'tanh'), Layer(3, 1, 'sigmoid')]
costs = [] # to plot graph

for epoch in range(epochs):
    A = x_train
    for layer in layers:
        A = layer.feedforward(A)

    cost = 1/m * np.sum(logloss(y_train, A))
    costs.append(cost)

    dA = d_logloss(y_train, A)
    for layer in reversed(layers):
        dA = layer.backprop(dA)
```

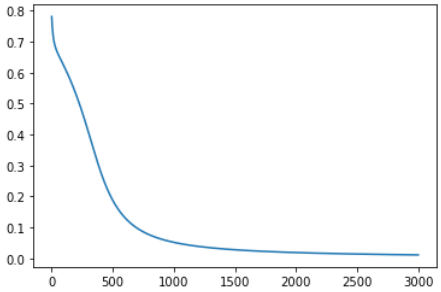
In [11]:
```
# predicting
A = np.array([[0], [0]]) # dim(=2) x m(=1)
for layer in layers:
    A = layer.feedforward(A)
print(A)
```

```
[[0.00805333]]
```

In [12]:

```
import matplotlib.pyplot as plt
plt.plot(range(epochs), costs)
```

Out[12]: [<matplotlib.lines.Line2D at 0x1b785b95b50>]



In [ ]: