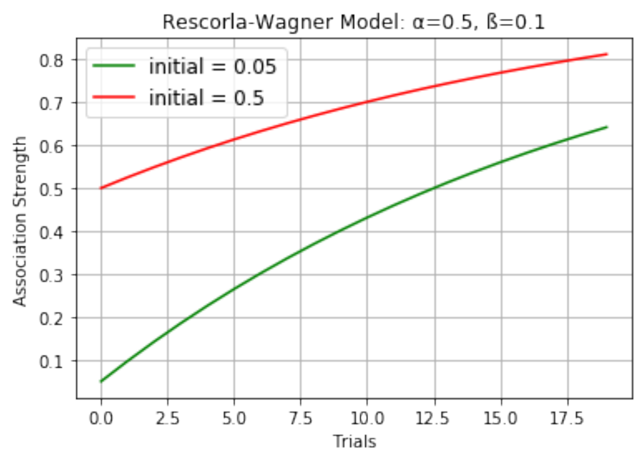
In [43]:

```python
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

#FUNCTION: reswag(num_trials, init_assoc)
#calculates Rescorla-Wagner equation for #'num_trials' trials
#'init_assoc' = initial association value
#returns a list 'strength' of all association strengths for 20 trials

def reswag(init_assoc, num_trials, alpha, beta):
    strength = []
    strength.append(init_assoc)
    counter = 0
    for counter in range(num_trials - 1):
        strength.append(strength[counter] + (alpha*beta*(1-strength[counter])))
    return strength

alpha = 0.5
beta = 0.1
x = np.arange(0, 20)
y1 = reswag(0.05, 20, alpha, beta)
y2 = reswag(0.5, 20, alpha, beta)

plt.plot(x, y1, color = 'green', label = 'initial = 0.05')
plt.plot(x, y2, color = 'red', label = 'initial = 0.5')
plt.title('Rescorla-Wagner Model: α=0.5, ß=0.1')
plt.xlabel('Trials')
plt.ylabel('Association Strength')
plt.legend(fontsize = 12)
plt.grid(True)
plt.show()
```



In [65]:

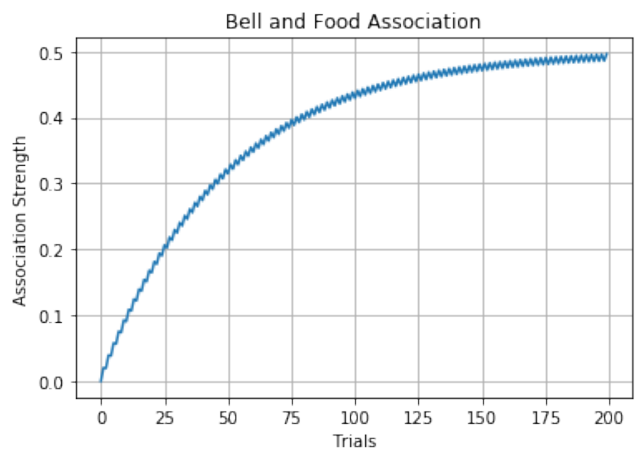
```python
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

#FUNCTION: reswag(num_trials, init_assoc)
#calculates Rescorla-Wagner equation for #'num_trials' trials
#'init_assoc' = initial association value
#returns a list 'strength' of all association strengths for 20 trials

def reswag(init_assoc, num_trials, alpha, beta):
    strength = []
    strength.append(init_assoc)
    counter = 0
    for counter in range(num_trials - 1):
        strength.append(strength[counter] + (alpha*beta*(1-strength[counter])))
    return strength

#When using an alpha value of 1.26 we see that the 13th trial Vx is
#greater than 0.8 if starting at an initial Vx of 0.0
alpha = 1.26
beta = 0.1
x = np.arange(0, 13)
y1 = reswag(0.0, 13, alpha, beta)
y1[12]
```

Out[65]:

```
0.8013277585362027
```

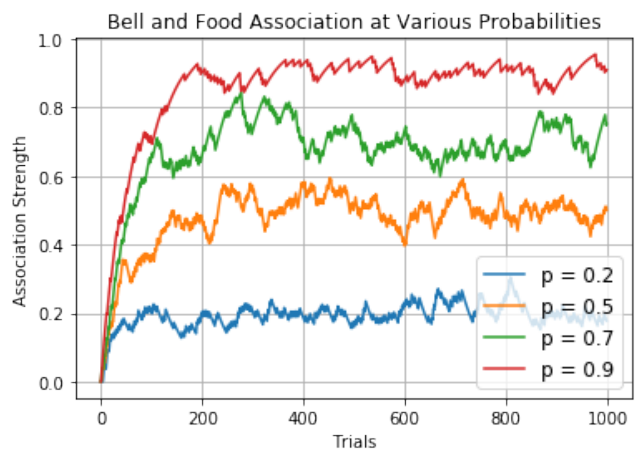
In [9]:

```python
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

#Vlight = 0.8
#Vbell (new association)
#alphabell = 0.2
#alphalight = 0.5
#beta = 0.1

def deltaVx_learning(alpha, beta, Vx1):
    deltaVx = alpha * beta * (1 - Vx1)
    return deltaVx

def deltaVx_extinction(alpha, beta, Vx1):
    deltaVx = -alpha * beta * Vx1
    return deltaVx

alphabell = 0.2
beta = 0.1
Vbell = 0.0
bellStrengths = [Vbell]

i = 0
while i < 199:
    if (i%2 == 0): #learning
        NewVbell = Vbell + deltaVx_learning(alphabell, beta, Vbell)
    else: #extinction
        NewVbell = Vbell + deltaVx_extinction(alphabell, beta, Vbell)
    i += 1
    bellStrengths.append(NewVbell)
    Vbell = NewVbell

plt.plot(np.arange(200), bellStrengths)
plt.title('Bell and Food Association')
plt.ylabel('Association Strength')
plt.xlabel('Trials')
plt.grid(True)

#In this question, starting at an original association strength of 0
#with a relatively low salience (abell = 0.2), it makes sense that the
#overall learning curve increases. Learning occurs faster than
#unlearning which is why the graph has an overall upwards trend.
```



In [40]:

```python
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

#Vlight = 0.8
#Vbell (new association)
#alphabell = 0.2
#alphalight = 0.5
#beta = 0.1

def deltaVx_learning(alpha, beta, Vx1):
    deltaVx = alpha * beta * (1 - Vx1)
    return deltaVx

def deltaVx_extinction(alpha, beta, Vx1):
    deltaVx = -alpha * beta * Vx1
    return deltaVx

def calc_strengths(alpha, beta, Vx1, prob):
    i = 0
    num = np.random.random()
    bell_strengths = [Vx1]
    while i < 999:
        if (num < prob): #learning
            NewVbell = Vx1 + deltaVx_learning(alphabell, beta, Vx1)
        else: #extinction
            NewVbell = Vx1 + deltaVx_extinction(alphabell, beta, Vx1)
        i += 1
        bell_strengths.append(NewVbell)
        Vx1 = NewVbell
        num = np.random.random()
    return bell_strengths

alphabell = 0.2
beta = 0.1
Vbell = 0.0
bellStrengths = [Vbell]
bellStrengths = calc_strengths(alphabell, beta, Vbell, 0.2)
plt.plot(np.arange(1000), bellStrengths, label = 'p = 0.2')

bellStrengths = [Vbell]
bellStrengths = calc_strengths(alphabell, beta, Vbell, 0.5)
plt.plot(np.arange(1000), bellStrengths, label = 'p = 0.5')

bellStrengths = [Vbell]
bellStrengths = calc_strengths(alphabell, beta, Vbell, 0.7)
plt.plot(np.arange(1000), bellStrengths, label = 'p = 0.7')

bellStrengths = [Vbell]
bellStrengths = calc_strengths(alphabell, beta, Vbell, 0.9)
plt.plot(np.arange(1000), bellStrengths, label = 'p = 0.9')

plt.title('Bell and Food Association at Various Probabilities')
plt.ylabel('Association Strength')
plt.xlabel('Trials')
plt.legend(fontsize = 12)
plt.grid(True)

#The trend which can be seen here is that each association strength is directly
#proportional to the probability of the organism learning. For example, if the organism
#learns 20% percent of the time, it will only ever become 20% optimal of its association
#strength or learn "20% of what the organism is capable of". In Marr's computational
#level, the organism is only as optimal as the probability in which it learns.
```



1. We think of salience and learning rate as different factors because, as human organisms ourselves, we associate learning rate as an intrinsic factor which is internal and therefore within our own capacity to control. In contrast, salience is seen as something which is external and therefore out of our control, making them seem like two very different factors in an experiment, despite the fact that they play the same role in the model. An experiment that would let you disentangle salience and learning rate would be to condition multiple types of organisms (a monkey, pig, cat, mouse and dog, for example) on a stimulus of a consistent salience. This way one could see how the association strength of each organism grew with time, and any variance in this growth from organism to organism would be due to a change in learning rate.