# Problem 4, Parts C-E: Stochastic Gradient Descent Visualization

In this Jupyter notebook, we visualize how SGD works. This visualization corresponds to parts C-E of question 4 in set 1.

Use this notebook to write your code for problem 4 parts C-E by filling in the sections marked `# TODO` and running all cells.

In [81]:
```
# Setup.

import numpy as np
import matplotlib.pyplot as plt
from IPython.display import HTML
import math

from sgd_helper import (
    generate_dataset1,
    generate_dataset2,
    plot_dataset,
    plot_loss_function,
    animate_convergence,
    animate_sgd_suite
)
```

## Problem 4C: Implementation of SGD

Fill in the loss, gradient, and SGD functions according to the guidelines given in the problem statement in order to perform SGD.

In [82]:
```
def loss(X, Y, w):
    '''
    Calculate the squared loss function.

    Inputs:
        X: A (N, D) shaped numpy array containing the data points.
        Y: A (N, ) shaped numpy array containing the (float) labels of the data po
ints.
        w: A (D, ) shaped numpy array containing the weight vector.

    Outputs:
        The loss evaluated with respect to X, Y, and w.
    '''
    predict = []
    for x in X:
        predict.append(np.inner(w, x))
    predict = np.asarray(predict)

    loss = 0
    for i in range(len(predict)):
        loss += (predict[i] - Y[i]) ** 2

    return loss

def gradient(x, y, w):
    '''
    Calculate the gradient of the loss function with  respect to
    a single point (x, y), and using weight vector w.

    Inputs:
        x: A (D, ) shaped numpy array containing a single data point.
        y: The float label for the data point.
        w: A (D, ) shaped numpy array containing the weight vector.

    Output:
        The gradient of the loss with respect to x, y, and w.
    '''

    #==============================================
    # TODO: Implement the gradient of the loss function.
    #==============================================
    grad = -2 * (y - np.inner(w, x)) * x
    return grad
```

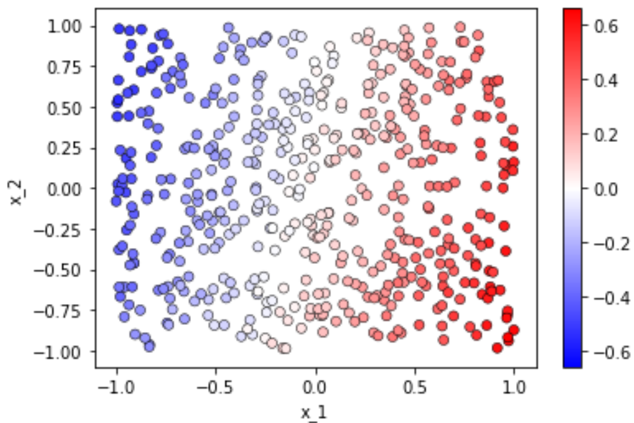
In [ ]:
```
def SGD(X, Y, w_start, eta, N_epochs):
    '''
    Perform SGD using dataset (X, Y), initial weight vector w_start,
    learning rate eta, and N_epochs epochs.

    Outputs:
        w: A (D, ) shaped array containing the final weight vector.
        losses: A (N_epochs, ) shaped array containing the losses from all iterati
ons.
    '''

    #==============================================
    # TODO: Implement the SGD algorithm.
    #==============================================

    totalLoss = []
    weights = w_start

    #start loss func (for some reason it won't work when i call the function?)
    #Python TypeError: 'list' object is not callable.
    #and
    #TypeError: 'numpy.float64' object is not callable
    predict = []

    for x in X:
        predict.append(np.inner(weights, x))
    predict = np.asarray(predict)

    loss = 0
    for i in range(len(predict)):
        loss += (predict[i] - Y[i]) ** 2
    #end loss func

    #totalLoss.append(loss)

    for n in range(N_epochs):
        for i in range(len(X)):
            g = gradient(X[i],Y[i],weights)
            weights -= eta * g

        #start loss func
        predict = []
        for x in X:
            predict.append(np.inner(weights, x))
        predict = np.asarray(predict)

        loss = 0
        assert(len(predict) == len(Y))
        for i in range(len(predict)):
            loss += (predict[i] - Y[i]) ** 2

        #currLoss = loss(X, Y, weights)
        totalLoss.append(loss)
        #print(str(n) + ": " + str(loss) + ", " + str(weights))

    return np.asarray(weights), np.asarray(totalLoss)
```

## Problem 4D: Visualization

## Dataset

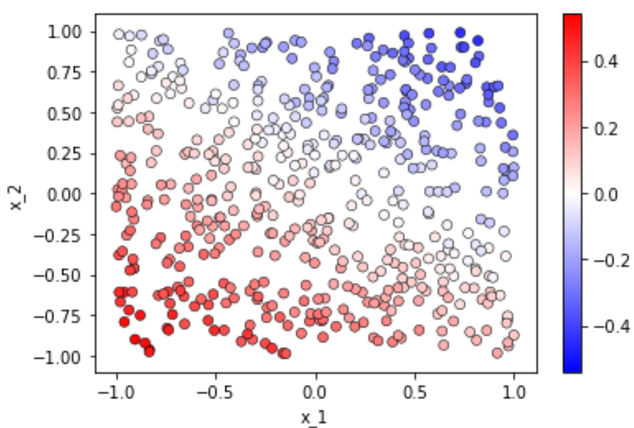
We'll start off by generating two simple 2-dimensional datasets. For simplicity we do not consider separate training and test sets.

In [83]:
```
X1, Y1 = generate_dataset1()
plot_dataset(X1, Y1)
```




Out[83]:
```
(<Figure size 432x288 with 2 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x11708fa90>)
```

In [84]:
```
X2, Y2 = generate_dataset2()
plot_dataset(X2, Y2)
```




Out[84]:
```
(<Figure size 432x288 with 2 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x1170e0750>)
```

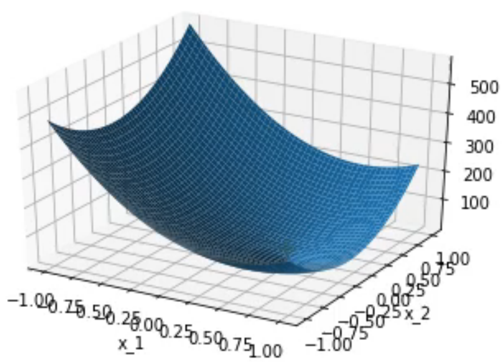
## SGD from a single point

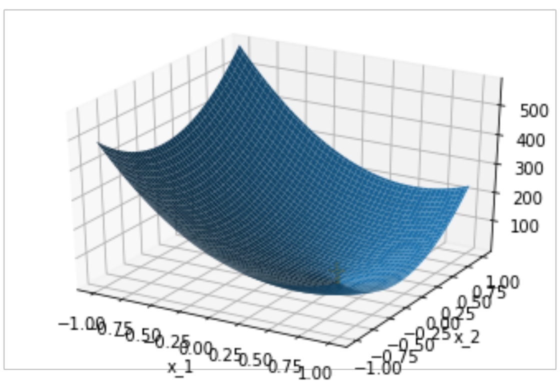
First, let's visualize SGD from a single starting point:

In [85]:
```
# Parameters to feed the SGD.
# <FR> changes the animation speed.
params = ({'w_start': [0.01, 0.01], 'eta': 0.00001},)
N_epochs = 1000
FR = 20

# Let's animate it!
anim = animate_sgd_suite(SGD, loss, X1, Y1, params, N_epochs, FR)
HTML(anim.to_html5_video())
```

```
Performing SGD with parameters {'w_start': [0.01, 0.01], 'eta': 1e-05} ...

Animating...
```

Out[85]:



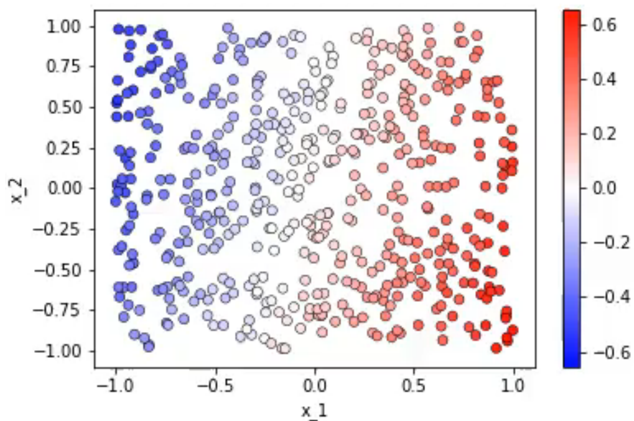



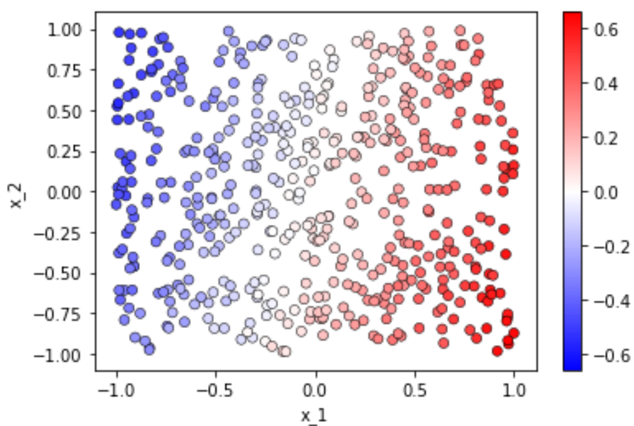

Let's view how the weights change as the algorithm converges:

In [86]:
```
# Parameters to feed the SGD.
params = ({'w_start': [0.01, 0.01], 'eta': 0.00001},)
N_epochs = 1000
FR = 20

# Let's do it!
W, _ = SGD(X1, Y1, params[0]['w_start'], params[0]['eta'], N_epochs)
anim = animate_convergence(X1, Y1, W, FR)
HTML(anim.to_html5_video())
```

```
Animating...
```

Out[86]:



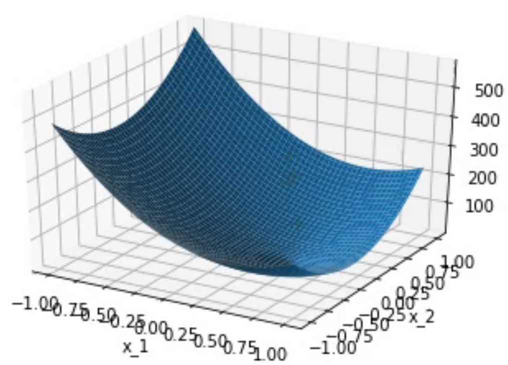

## SGD from multiple points

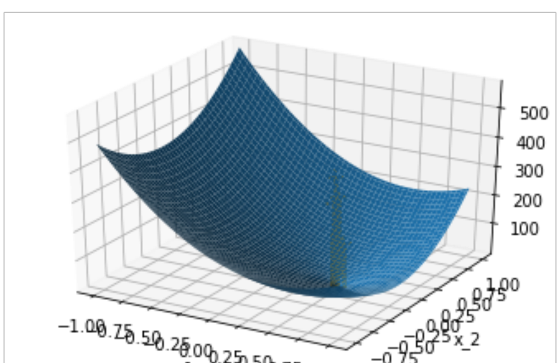
Now, let's visualize SGD from multiple arbitrary starting points:

In [87]:
```
# Parameters to feed the SGD.
# Here, we specify each different set of initializations as a dictionary.
params = (
    {'w_start': [-0.8, -0.3], 'eta': 0.00001},
    {'w_start': [-0.9, 0.4], 'eta': 0.00001},
    {'w_start': [-0.4, 0.9], 'eta': 0.00001},
    {'w_start': [0.8, 0.8], 'eta': 0.00001},
)
N_epochs = 1000
FR = 20

# Let's go!
anim = animate_sgd_suite(SGD, loss, X1, Y1, params, N_epochs, FR)
HTML(anim.to_html5_video())
```

```
Performing SGD with parameters {'w_start': [-0.8, -0.3], 'eta': 1e-05} ...
Performing SGD with parameters {'w_start': [-0.9, 0.4], 'eta': 1e-05} ...
Performing SGD with parameters {'w_start': [-0.4, 0.9], 'eta': 1e-05} ...
Performing SGD with parameters {'w_start': [0.8, 0.8], 'eta': 1e-05} ...

Animating...
```

Out[87]:



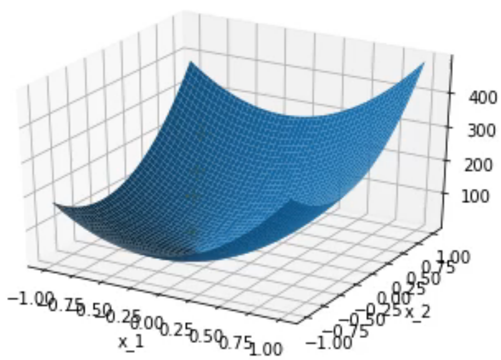

Let's do the same thing but with a different dataset:

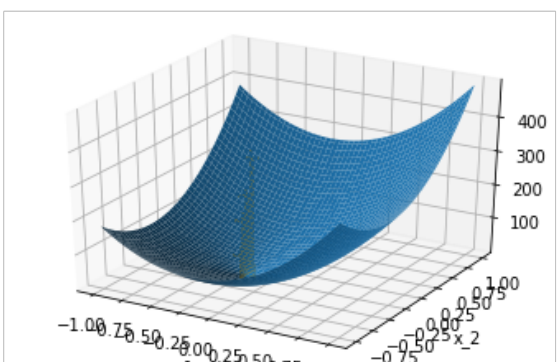
In [88]:

```
# Parameters to feed the SGD.
params = (
    {'w_start': [-0.8, -0.3], 'eta': 0.00001},
    {'w_start': [-0.9, 0.4], 'eta': 0.00001},
    {'w_start': [-0.4, 0.9], 'eta': 0.00001},
    {'w_start': [0.8, 0.8], 'eta': 0.00001},
)
N_epochs = 1000
FR = 20

# Animate!
anim = animate_sgd_suite(SGD, loss, X2, Y2, params, N_epochs, FR)
HTML(anim.to_html5_video())
```

```
Performing SGD with parameters {'w_start': [-0.8, -0.3], 'eta': 1e-05} ...
Performing SGD with parameters {'w_start': [-0.9, 0.4], 'eta': 1e-05} ...
Performing SGD with parameters {'w_start': [-0.4, 0.9], 'eta': 1e-05} ...
Performing SGD with parameters {'w_start': [0.8, 0.8], 'eta': 1e-05} ...

Animating...
```

Out[88]:




## Problem 4E: SGD with different step sizes

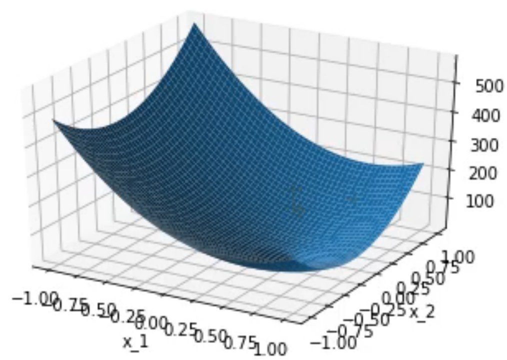
Now, let's visualize SGD with different step sizes (eta):

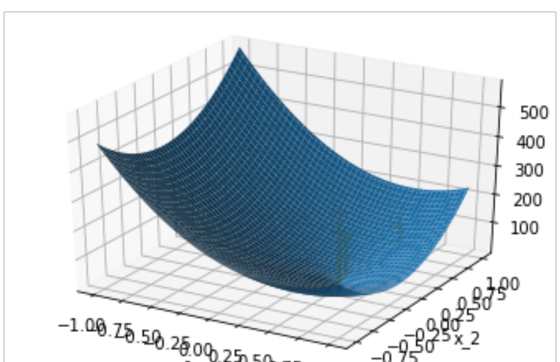
(For ease of visualization: the trajectories are ordered from left to right by increasing eta value. Also, note that we use smaller values of N_epochs and FR here for easier visualization.)

In [89]:
```
# Parameters to feed the SGD.
params = (
    {'w_start': [0.7, 0.8], 'eta': 0.00001},
    { 'w_start': [0.2, 0.8], 'eta': 0.00005},
    {'w_start': [-0.2, 0.7], 'eta': 0.0001},
    {'w_start': [-0.6, 0.6], 'eta': 0.0002},
)
N_epochs = 100
FR = 2

# Go!
anim = animate_sgd_suite(SGD, loss, X1, Y1, params, N_epochs, FR, ms=2)
HTML(anim.to_html5_video())
```

```
Performing SGD with parameters {'w_start': [0.7, 0.8], 'eta': 1e-05} ...
Performing SGD with parameters {'w_start': [0.2, 0.8], 'eta': 5e-05} ...
Performing SGD with parameters {'w_start': [-0.2, 0.7], 'eta': 0.0001} ...
Performing SGD with parameters {'w_start': [-0.6, 0.6], 'eta': 0.0002} ...

Animating...
```

Out[89]:



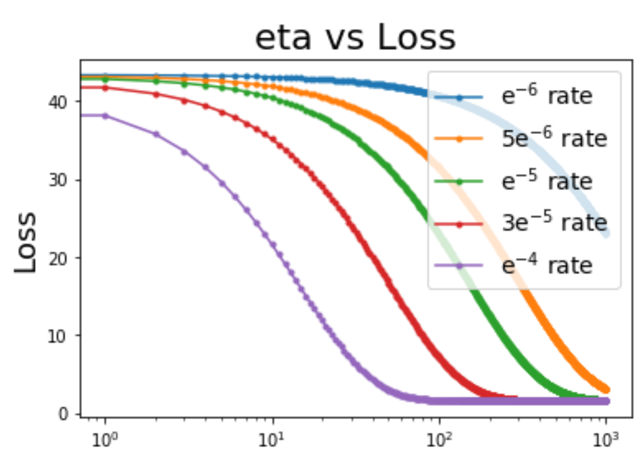




### Plotting SGD Convergence

Let's visualize the difference in convergence rates a different way. Plot the loss with respect to epoch (iteration) number for each value of eta on the same graph.

In [90]:
```
'''Plotting SGD convergence'''

#==============================================
# TODO: For the given learning rates, plot the
# loss for each epoch.
#==============================================

eta_vals = [1e-6, 5e-6, 1e-5, 3e-5, 1e-4]
w_start = [0.01, 0.01]
N_epochs = 1000

weights = []
loss = []

for eta in eta_vals:
    initial = [0.001, 0.001, 0.001, 0.001, 0.001]
    finalWeights, allLoss = SGD(X1, Y1, w_start, eta, N_epochs)
    weights.append(finalWeights)
    loss.append(allLoss)

fig = plt.figure()
x = epochs
plt.title('eta vs Loss', fontsize = 22)
plt.plot(loss[0], marker = '.')
plt.plot(loss[1], marker = '.')
plt.plot(loss[2], marker = '.')
plt.plot(loss[3], marker = '.')
plt.plot(loss[4], marker = '.')
plt.legend(('e$^{-6}$ rate', '5e$^{-6}$ rate', 'e$^{-5}$ rate', '3e$^{-5}$ rate',
'e$^{-4}$ rate'), loc = 'upper right', fontsize = 14)
plt.xscale('log')
plt.ylabel('Loss', fontsize = 18)
```

Out[90]: `Text(0, 0.5, 'Loss')`




Clearly, a big step size results in fast convergence! Why don't we just set eta to a really big value, then? Say, eta=1?

(Again, note that the FR is lower for this animation.)

In [91]:
```
# Parameters to feed the SGD.
params = ({'w_start': [0.01, 0.01], 'eta': 1},)
N_epochs = 100
FR = 2

# Voila!
anim = animate_sgd_suite(SGD, loss, X1, Y1, params, N_epochs, FR, ms=2)
HTML(anim.to_html5_video())
```

```
Performing SGD with parameters {'w_start': [0.01, 0.01], 'eta': 1} ...

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-91-6f8ee3a8be68> in <module>
      5 
      6 # Voila!
----> 7 anim = animate_sgd_suite(SGD, loss, X1, Y1, params, N_epochs, FR, ms=2)
      8 HTML(anim.to_html5_video())

~/Downloads/CS155_SET1/sgd_helper.py in animate_sgd_suite(SGD, loss, X, Y, param
s, N_epochs, FR, ms)
    121 
    122     # Get the loss grid and plot it.
--> 123     w_grid, loss_grid = get_loss_grid((-1, 1, 100), (-1, 1, 100), X, Y,
loss)
    124     fig, ax = plot_loss_function(w_grid[0], w_grid[1], loss_grid)
    125 

~/Downloads/CS155_SET1/sgd_helper.py in get_loss_grid(x_params, y_params, X, Y,
loss)
     77         for j in range(len(loss_grid[0])):
     78             w = np.array([w_grid[0][i, j], w_grid[1][i, j]])
---> 79             loss_grid[i, j] = loss(X, Y, w)
     80 
     81     return w_grid, loss_grid

TypeError: 'list' object is not callable
```

Just for fun, let's try eta=10 as well. What happens? (Hint: look at W)

In [ ]:
```
# Parameters to feed the SGD.
w_start = [0.01, 0.01]
eta = 10
N_epochs = 100

# Presto!
W, losses = SGD(X1, Y1, w_start, eta, N_epochs)
```

## Extra Visualization (not part of the homework problem)

One final visualization! What happens if the loss function has multiple optima?

In [ ]:

```
# Import different SGD & loss functions.
# In particular, the loss function has multiple optima.
from sgd_multiopt_helper import SGD, loss

# Parameters to feed the SGD.
params = (
    {'w_start': [0.9, 0.9], 'eta': 0.01},
    { 'w_start': [0.0, 0.0], 'eta': 0.01},
    {'w_start': [-0.8, 0.6], 'eta': 0.01},
    {'w_start': [-0.8, -0.6], 'eta': 0.01},
    {'w_start': [-0.4, -0.3], 'eta': 0.01},
)
N_epochs = 100
FR = 2

# One more time!
anim = animate_sgd_suite(SGD, loss, X1, Y1, params, N_epochs, FR, ms=2)
HTML(anim.to_html5_video())
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: