HW1 - Abhi Kamboj

Tuesday, September 15, 2020 3:59 PM

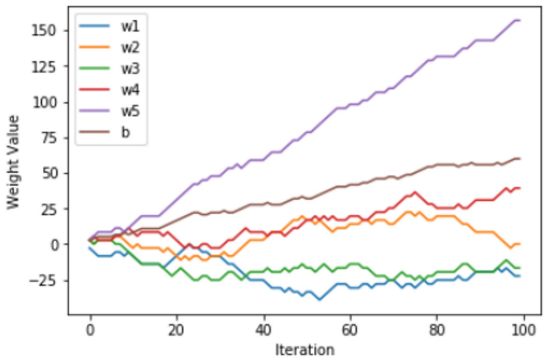
Problem 1:

1.

v	w	x	y	z	f
1	1	1	1	1	1
1	1	1	1	-1	-1
1	1	1	-1	1	-1
1	1	1	-1	-1	-1
1	1	-1	1	1	-1
1	1	-1	1	-1	-1
1	1	-1	-1	1	-1
1	1	-1	-1	-1	1
1	-1	1	1	1	-1
1	-1	1	1	-1	-1
1	-1	1	-1	1	-1
1	-1	1	-1	-1	1
1	-1	-1	1	1	-1
1	-1	-1	1	-1	-1
1	-1	-1	-1	1	-1
1	-1	-1	-1	-1	1
-1	1	1	1	1	-1
-1	1	1	1	-1	-1
-1	1	1	-1	1	-1
-1	1	1	-1	-1	1
-1	1	-1	1	1	-1
-1	1	-1	1	-1	-1
-1	1	-1	-1	1	-1
-1	1	-1	-1	-1	1
-1	-1	1	1	1	1
-1	-1	1	1	-1	-1
-1	-1	1	-1	1	-1
-1	-1	1	-1	-1	-1
-1	-1	-1	1	1	-1
-1	-1	-1	1	-1	-1
-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	1

1.2

Weights and bias values over training iterations for Adaline:



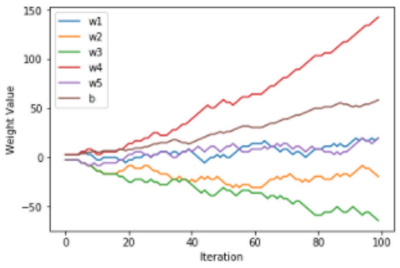
The function accurately classified all 32 possible inputs. Here is the Code

```
#truth_table:
arr=np.array([
[1,1,1,1,1],
[1,1,1,1,-1],
[1,1,1,-1,1],
[1,1,1,-1,-1],
[1,1,-1,1,1],
[1,1,-1,1,-1],
[1,1,-1,-1,1],
[1,1,-1,-1,-1],
[1,-1,1,1,1],
[1,-1,1,1,-1],
[1,-1,1,-1,1],
[1,-1,1,-1,-1],
[1,-1,-1,1,1],
[1,-1,-1,1,-1],
[1,-1,-1,-1,1],
[1,-1,-1,-1,-1],
[-1,1,1,1,1],
[-1,1,1,1,-1],
[-1,1,1,-1,1],
[-1,1,1,-1,-1],
[-1,1,-1,1,1],
[-1,1,-1,1,-1],
[-1,1,-1,-1,1],
[-1,1,-1,-1,-1],
[-1,-1,1,1,1],
[-1,-1,1,1,-1],
[-1,-1,1,-1,1],
[-1,-1,1,-1,-1],
[-1,-1,-1,1,1],
[-1,-1,-1,1,-1],
[-1,-1,-1,-1,1],
[-1,-1,-1,-1,-1],
])

def f(arr):
    def xor(a,b):
        if a==b:
            return 1
        elif a==-b:
            return -1

    def or_(a,b):
        if (a==b) and (b==1):
            return 1
        else:
            return -1
    v,w,x,y,z = arr
    return or_(xor(or_(xor(v,w),x),y), xor(y,z))
```

In [13]:

```
# x is (v w x y z) data points
# n samples
n=100
xs = arr

xs = np.random.randint(0,2,(n,5))
xs[ xs <1]=-1

y = np.apply_along_axis(f, 1, xs)
w = np.zeros(5)
b = 1
eta = .7
evol = []
errors=[]

for i,x in enumerate(xs):
    #print(i,x,y[i])
    y_hat = np.sign(np.dot(x,w)+b)
    #print("y_hat",y_hat)
    e =y[i]-y_hat
    errors.append(e)
    w = w - 2*eta*e*x
    b = b - eta*e
    evol.append(np.append(w,b))

#print(evol)
#print(errors)
plt.figure()
plt.xlabel("Iteration")
plt.ylabel("Weight Value")
plt.plot(range(len(evol)),evol)

plt.legend(["w1","w2","w3","w4","w5","b"])

y_true = y = np.apply_along_axis(f, 1, arr)

print("Number of correct:", np.count_nonzero(np.matmul(arr,w)-y_true))
```

```
Number of correct: 32
```



Proble

2.1

For

so

slope $= \frac{1}{2} - 4$

so $\int_{t_q}^{t_{q+1}} x f(x)\,dx = 2\int_{t_q}^{t_{q+1}} \frac{1}{8}x^2 + \frac{1}{4}x = 2\left[\frac{x^3}{24} + \frac{x^2}{8}\right]_{t_q}^{t_{q+1}} = 2\frac{x^2}{8}\left(\frac{x}{3}+1\right)\Big|_{t_q}^{t_{q+1}}$

$\int_{t_q}^{t_{q+1}} f(x)\,dx = 2\int_{t_q}^{t_{q+1}} \frac{1}{8}x + \frac{1}{4} = 2\left[\frac{x^2}{16} + \frac{x}{4}\right]_{t_q}^{t_{q+1}} = 2\frac{x}{4}\left(\frac{x}{4}+1\right)\Big|_{t_q}^{t_{q+1}}$

$$r_q = \frac{\frac{(t_{q+1})^2}{8}\left(\frac{t_{q+1}}{3}+1\right) - \frac{t_q^2}{8}\left(\frac{t_q}{3}+1\right)}{\frac{t_{q+1}}{4}\left(\frac{t_{q+1}}{4}+1\right) - \frac{t_q}{4}\left(\frac{t_q}{4}+1\right)}$$

for $x > 0$

slope $= -\frac{1}{2}$ $\quad f(x) = -\frac{1}{4}x + \frac{1}{2}$

so $\int_{t_q}^{t_{q+1}} x F(x)\,dx = 2\int_{t_q}^{t_{q+1}} -\frac{1}{8}x^2 + \frac{1}{4}x\,dx = 2\left[-\frac{x^3}{24} + \frac{x^2}{8}\right]_{t_q}^{t_{q+1}} = 2\frac{x^2}{8}\left(\frac{-x}{3}+1\right)\Big|_{t_q}^{t_{q+1}}$

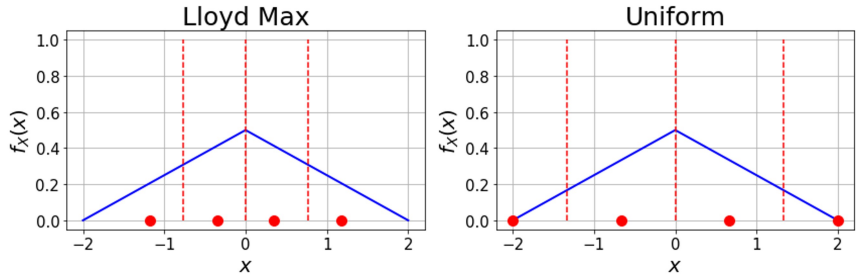
$\int_{t_q}^{t_{q+1}} f(x)\,dx = 2\int_{t_q}^{t_{q+1}} -\frac{1}{8}x + \frac{1}{4}\,dx = \frac{2}{4}\left(\frac{-x}{4}+1\right)\Big|_{t_q}^{t_{q+1}}$

$$r_q = \frac{\frac{t_{q+1}^2}{8}\left(-\frac{t_{q+1}}{3}+1\right) - \frac{t_q^2}{8}\left(-\frac{t_q}{3}+1\right)}{\frac{t_{q+1}}{4}\left(-\frac{t_{q+1}}{4}+1\right) - \frac{t_q}{4}\left(-\frac{t_q}{4}+1\right)}$$

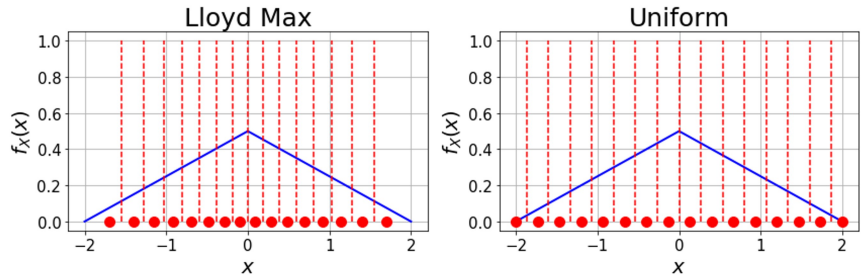
else:

$$r_q = \frac{\int_{t_q}^{0} x f(x<0) + \int_{0}^{t_{q+1}} x f(x>0)}{\int_{t_q}^{0} f(x<0) + \int_{0}^{t_{q+1}} f(x>0)} = \frac{-\frac{t_q^2}{8}\left(\frac{t_q}{3}+1\right) + \frac{t_{q+1}^2}{8}\left(1-\frac{t_{q+1}}{3}\right)}{\frac{t_q}{4}\left(\frac{t_q}{4}+1\right) + \frac{t_{q+1}}{4}\left(1-\frac{t_{q+1}}{4}\right)}$$

2.2




```
4 Levels
uniform levels: [-2.         -0.66666667  0.66666667  2.        ]
Llyod-Max levels: [-1.17595468 -0.35190936  0.35190936  1.17595468]
```



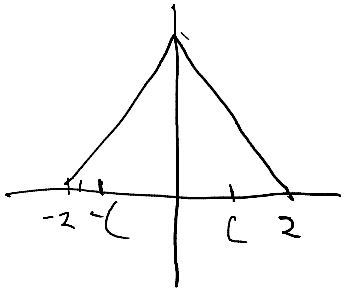

```
16 Levels
uniform levels: [-2.         -1.73333333 -1.46666667 -1.2        -0.93333333 -0.66666667
 -0.4        -0.13333333  0.13333333  0.4         0.66666667  0.93333333
  1.2         1.46666667  1.73333333  2.        ]
Llyod-Max levels: [-1.70116036 -1.40202869 -1.14591296 -0.91240506 -0.69362234 -0.48555369
 -0.2858983  -0.09325654  0.09325654  0.2858983   0.48555369  0.69362234
  0.91240506  1.14591296  1.40202869  1.70116036]
```

2.3 Empirically evaluated SQNRS:

```
4 Quantization Levels
uniform quantizer sqnr: 6.017483120576058
lloyd max quantizer sqnr: 10.333089985685175

16 Quantization Levels
uniform quantizer sqnr: 20.718238604722856
lloyd max quantizer sqnr: 21.675219329813697
```

2.5

$$SQNR_y = 10 \log_{10}\left(\frac{\sigma_x^2}{\sigma_{qx}^2}\right)$$



$$\Delta = \frac{2C}{2B_x}$$

$$\sigma_{q_x}^2 = \Delta^2 \quad \_ \; 4r^2 / 11 \quad r^2$$

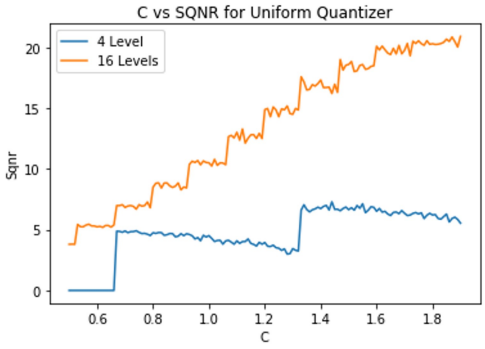
$$\sigma_{q_x}^2 = \frac{\Delta^2}{12} = \frac{4C^2}{2^{2B_x}}\left(\frac{1}{12}\right) = \frac{C^2}{3(2^{2B_x})}$$

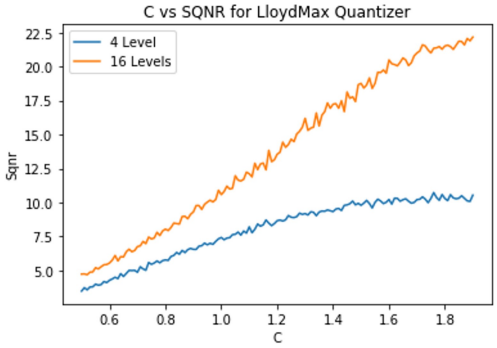
$$\frac{\sigma_x^2}{\sigma_{q_x}^2} = \frac{\sigma_x^2}{C^2}(3)2^{2B_x}$$

$$SQNR = 10\log\left(\frac{\sigma_x^2}{\sigma_{q_x}^2}\right) = \left(\log 3 + 2B_x \log 2 - \log\frac{C^2}{\sigma_x^2}\right)10$$

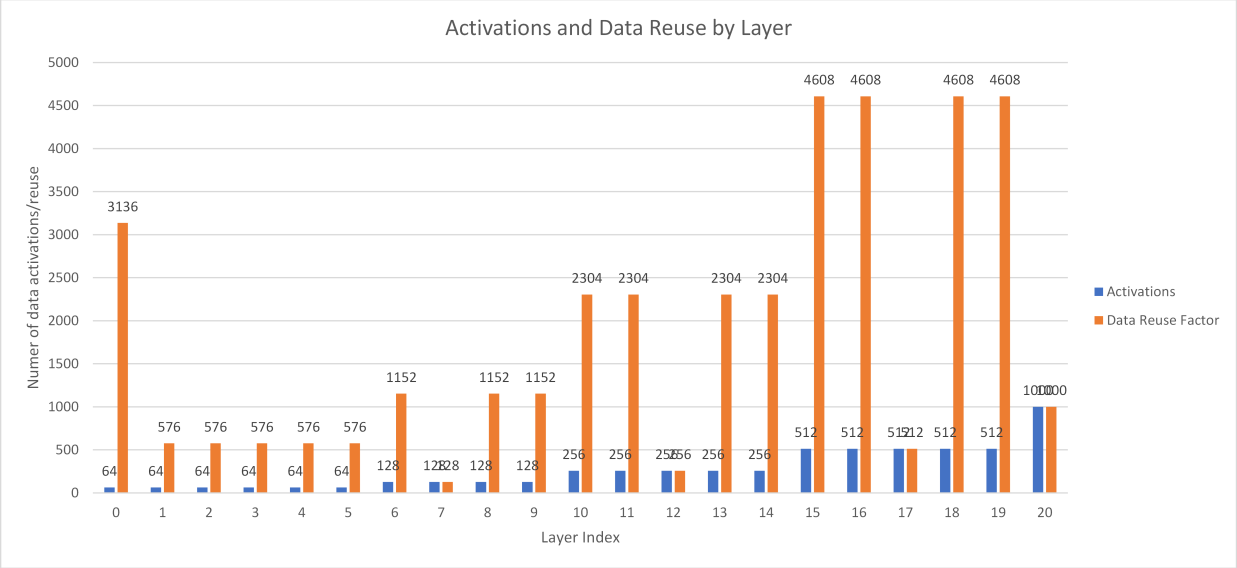
$$\boxed{SQNR = 4.78 + 6B_x - 20\frac{C}{\sigma_x}}$$

↑ For uniform distribution

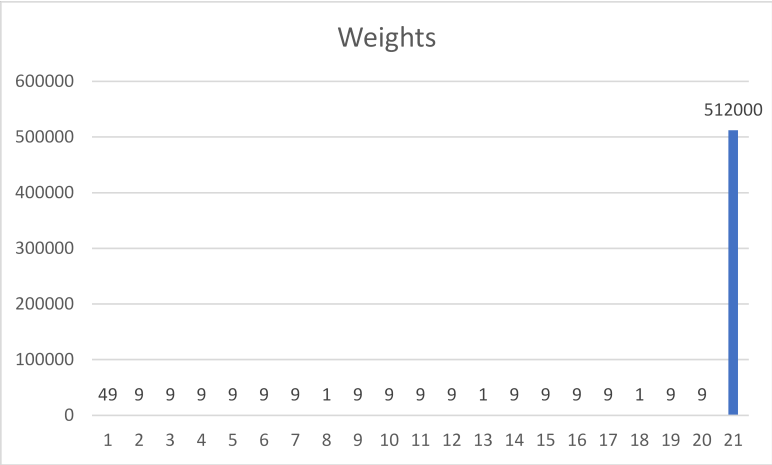


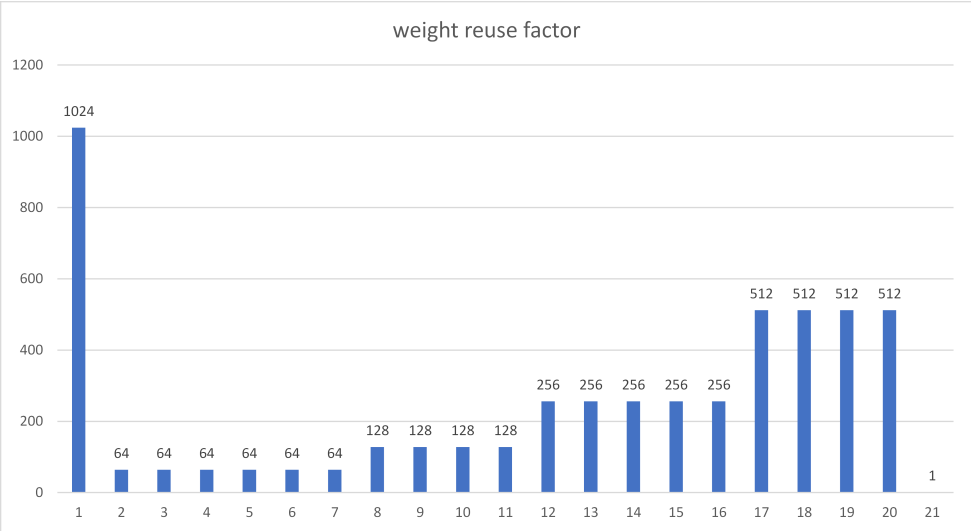


Problem 3:

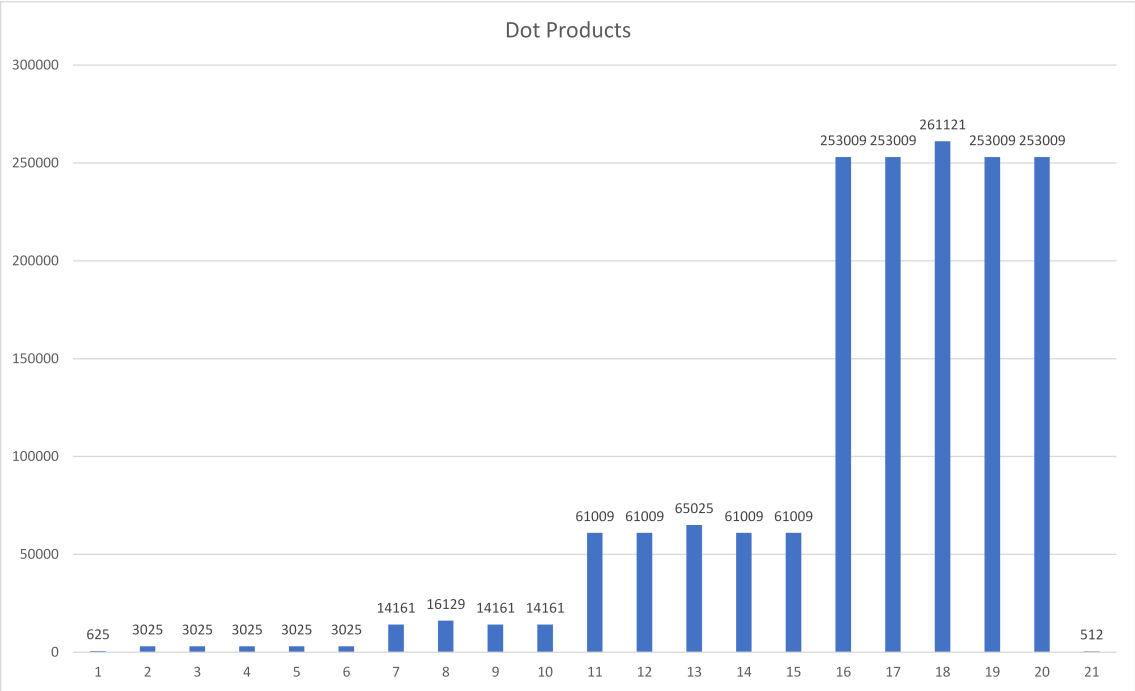


Total Activations: 5736

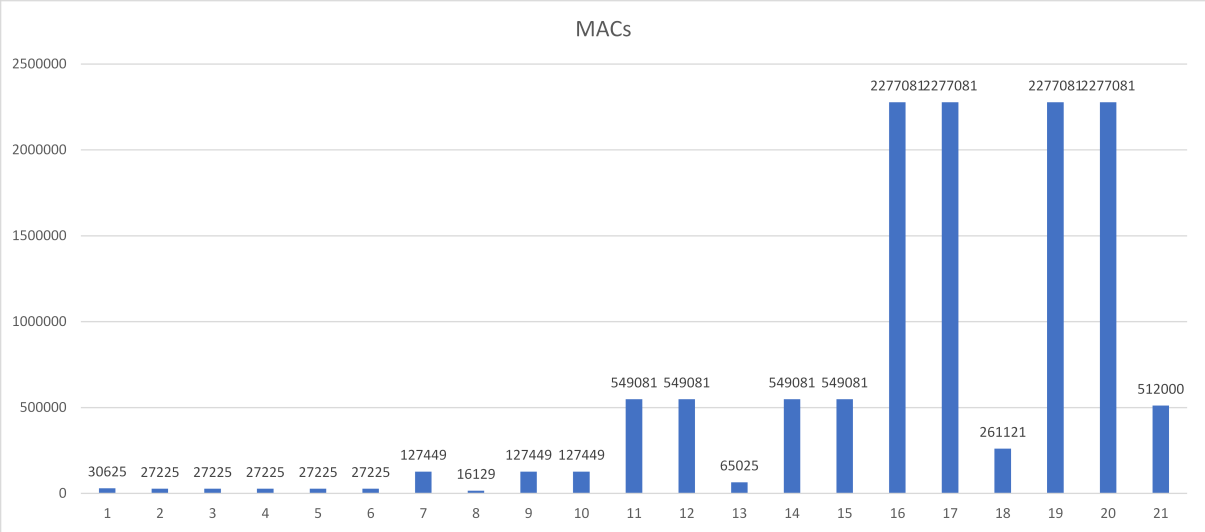




Total Number of Weights: 512196



Total Dot Products: 1,657,092



Total MACs: 12,708,020

3.5 Assuming activations and computations happen in place, we have 5736/1024 = 5.60 MB
3.6 E_dec = .5*12708020=6354010 pJ = 6.4 uJ (micro Joules)
3.7 MAC/N*10^-9 = 30 so we have N = 12708020/30/10^-9= 4.2360E14

Problem 4:

4.1
This neural network looks like a fairly standard feedforward deep convolutional neural network which has 3 input channels. It actually looks a lot like Alexnet. It has a feature extractor containing 5 sets of a convolutional layer followed by Relu activation and maxpooling. Then it has a classifier with a dropout layer, 2 linear (fully connected layers) with a dropout layer between them and 2 Relu activations after each, then ending with a final linear layer outputting the classifications. I can only find 2 differences between this network and Alexnet:

a. The first convolutional layer has a kernel_size of 3x3 instead of 11x11
b. The first linear layer (the weights of the network) has a size 4096x4096 instead of 9216x9216

The dataset is the CIFAR-10 Dataset that has small images of objects and animals. Each image is 32x32x3, and there are about 50k training samples and 10k test samples. It is a good midlevel training set for validating ideas and general insights and is used quite a b it in academic research. Typical accuracy is about 90%, good is 95% and state of the art is 98% on this dataset.

4.2
The accuracy of the model was determined to be 86.24%:

```
Evaluating Batch[ 20/100]       Acc@1  86.00 ( 86.30)
Evaluating Batch[ 40/100]       Acc@1  87.00 ( 86.33)
Evaluating Batch[ 60/100]       Acc@1  87.00 ( 86.05)
Evaluating Batch[ 80/100]       Acc@1  89.00 ( 86.14)
Evaluating Batch[100/100]       Acc@1  86.00 ( 86.24)
 * Acc@1 86.240
```

Problem 5:

5.1 Empirical MSE estimate using 1000 samples: 0.5145843504024726
5.2 Optimal Weights: [-0.63856452 -0.3524288 -0.19029405]
5.3 MSE: 0.3587265144264689

Here is my code:

Question 5

In [105]:
```
n_sample = 1000

gn = [np.random.normal(),np.random.normal(),np.random.normal()]
data = []
for idx in range(3,n_sample+3):
    g = np.random.normal()
    gn.append(g)
    data.append(.1*g+.5*gn[idx-1]-.5*gn[idx-2]+.1*gn[idx-3]) # x_[n] = u_[n] + 0.8*x_[n-1]
```

In [106]:
```
w = np.array([.4, -.1, .02])
x = data[0:3]
for idx in range(3,n_sample):
    x.append(np.dot(x[-3:],w))
print("Emperical MSE estimate using 1000 samples: ", np.sum(np.square(np.array(data) - np.array(x)))/n_sample)
```

```
Emperical MSE estimate using 1000 samples:  0.5145843504024726
```

In [107]:
```
X = []
y = []
COV_X = []
for three_sample_before, two_sample_before, one_sample_before, curr in zip(data[0:-3:], data[1:-2:], data[2:-1:],data[3::]):
    vector1 = np.array([one_sample_before, two_sample_before, three_sample_before])
    vector2 = np.array([curr, one_sample_before, two_sample_before]) # for R
    X.append(vector1)
    y.append(curr)
    COV_X.append(vector2)
X = np.vstack(X)
y = np.array(y)
COV_X = np.vstack(COV_X)
```

In [108]:
```
R = np.cov(COV_X.T)
p = y.dot(X)/len(y)
w = np.linalg.inv(R).dot(p)
prediction = X.dot(w)
print("Optimal Weights:",w)
error = y.dot(y)/n_sample - p.dot(w)
print("MSE:",error)
```

```
Optimal Weights: [-0.63856452 -0.3524288  -0.19029405]
MSE: 0.3587265144264689
```