solution02

November 10, 2020

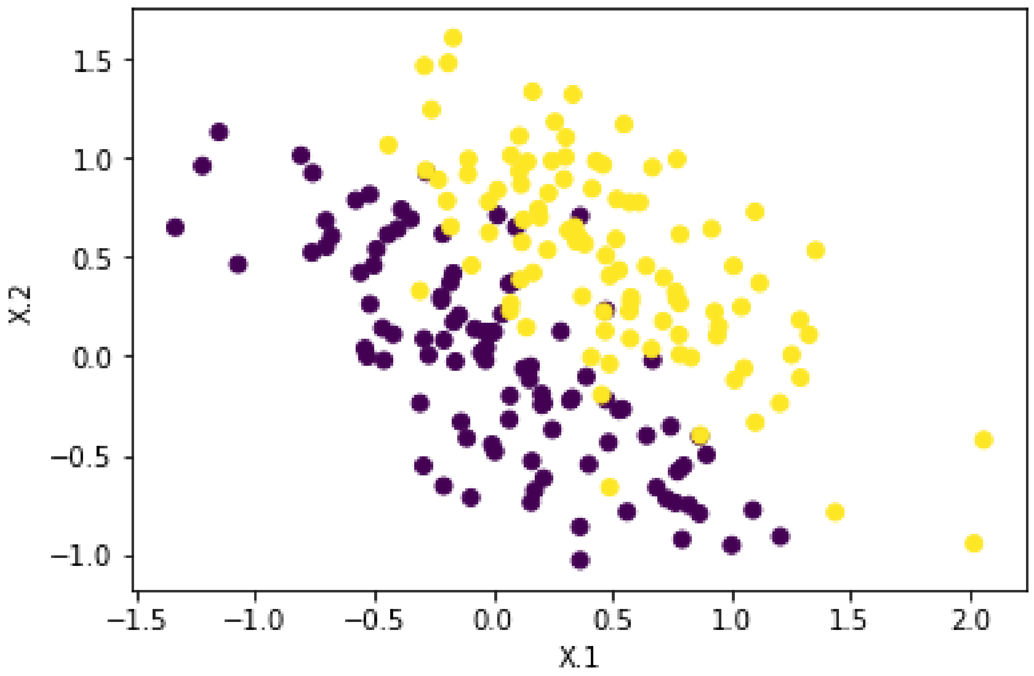
Exercise Sheet 2 **Connectionist Neurons and Multi Layer Perceptrons**

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

def sgn(x):
    x[x >= 0] = 1
    x[x < 0] = 0

def SLCN(X,W,t):
    if np.matmul(W.T,X)-t < 0:
        return 0
    else:
        return 1
```

2.1: Connectionist Neuron

(a)

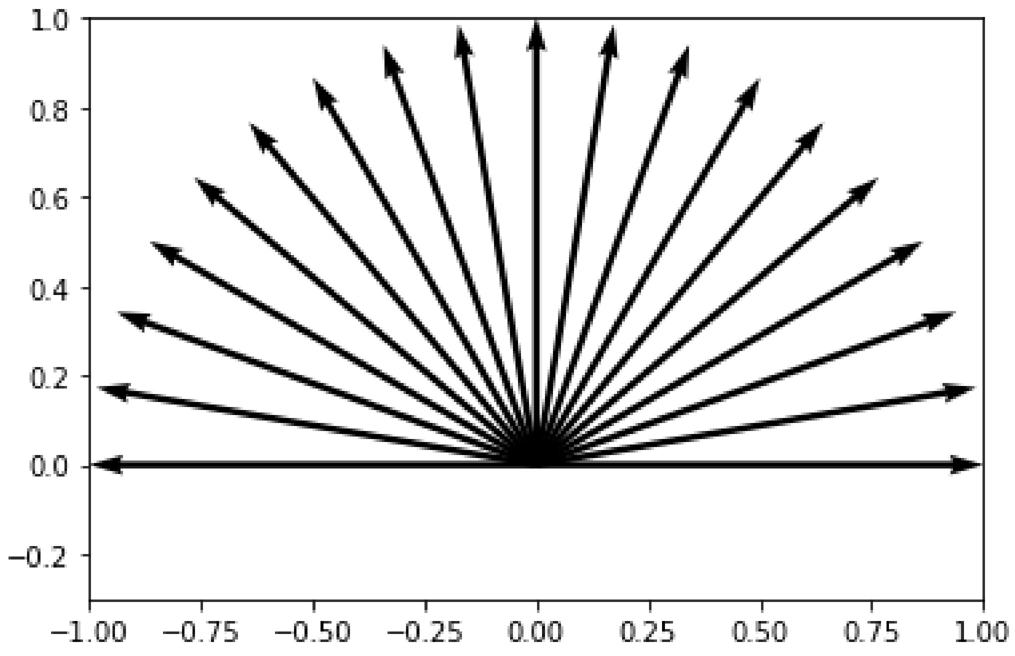
[2]:
```
aData = np.genfromtxt('applesOranges.csv',
        delimiter = ',',skip_header=1) # First row is the header
plt.figure()
plt.scatter(aData[:,0],aData[:,1],c=(aData[:,2]))
plt.xlabel('X.1')
plt.ylabel('X.2')
plt.show()
```



(b), (c)

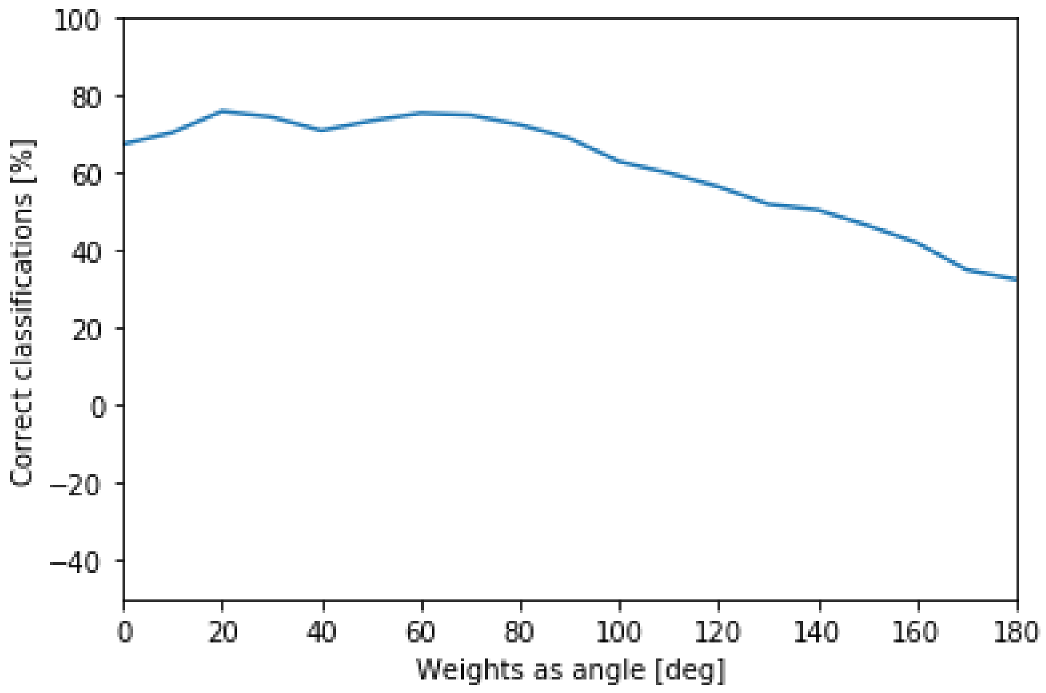
```
[3]: origin = [0], [0]
     a_deg = np.linspace(0,180,19,endpoint=True)
     a = np.deg2rad(a_deg)
     W = np.vstack(((np.cos(a)), (np.sin(a)))).T
     plt.figure()
     plt.quiver(*origin, W[:,0], W[:,1], scale=2)
     plt.xlim(-1, 1)
     plt.ylim(-0.3, 1)
     plt.show
```

```
[3]: <function matplotlib.pyplot.show(*args, **kw)>
```



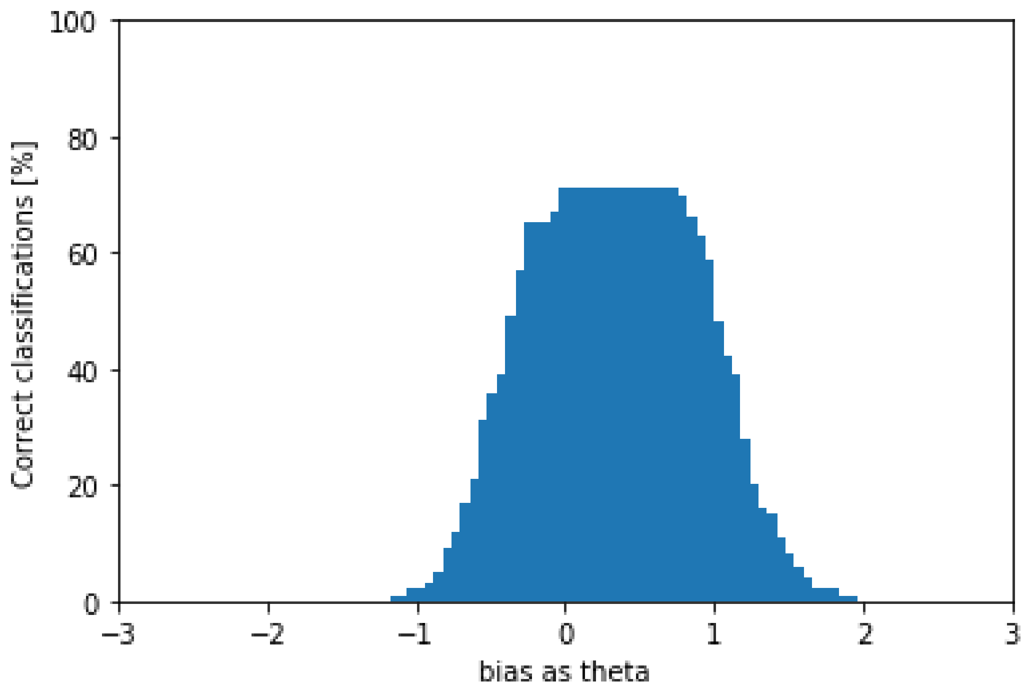
[4]:
```
Y = np.vstack(([a_deg],[np.zeros(len(a))])).T
theta = 0
for i in range(np.shape(W)[0]):
    for j in range(np.shape(aData)[0]):
        if SLCN(aData[j,0:2],W[i],theta) == aData[j,2]:
            Y[i,1] += 1
print('At %s degree and theta of %f the p is maximum of %f%%\n'
      'for %d correct classifications!'
      %(Y[np.argmax(Y[:,1]),0],theta,np.max(Y[:,1])/2,np.max(Y[:,1])))
plt.figure()
plt.plot(a_deg,Y[:,1]/2)
plt.xlim(0, 180)
plt.ylim(-50, 100)
plt.xlabel('Weights as angle [deg]')
plt.ylabel('Correct classifications [%]')
plt.show()
```

```
At 20.0 degree and theta of 0.000000 the p is maximum of 76.000000%
for 152 correct classifications!
```



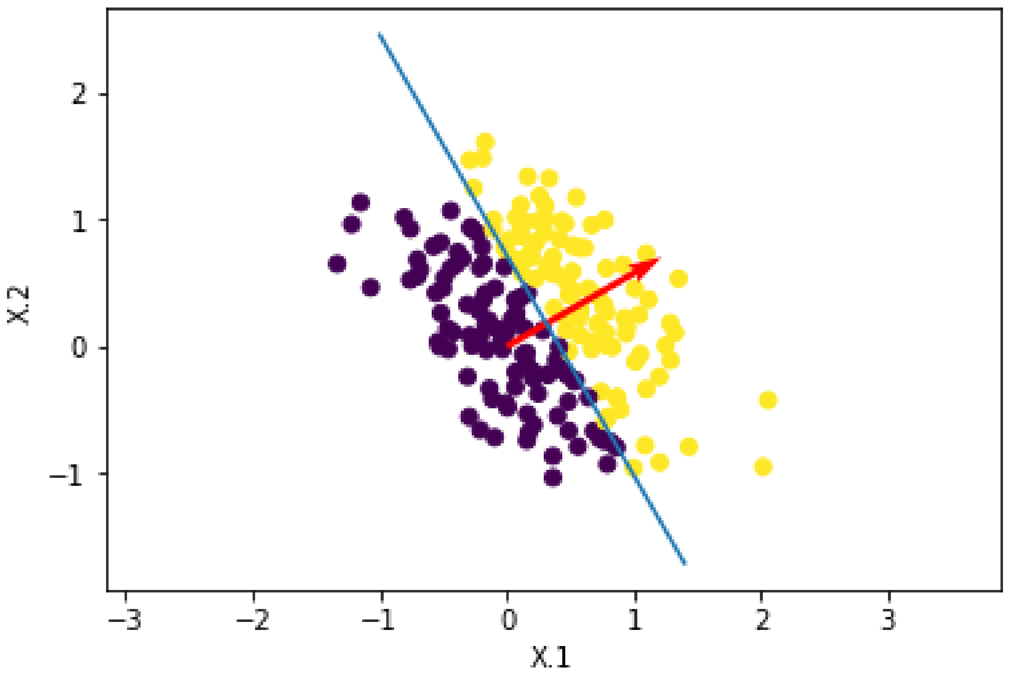
```
[5]: theta = np.linspace(-3,3,101,endpoint=True)
     Z = np.zeros(len(theta))
     for i in range(len(theta)):
         for j in range(np.shape(aData)[0]):
             if SLCN(aData[j,0:2],W[3],theta[i]) == aData[j,2]:
                 Z[i] += 1
             else:
                 Z[i] -= 1
     print('At 20 degree and theta of %f the p is maximum of %f%% \n'
           'for %d correct classifications!'
           %(theta[np.argmax(Z)],np.max(Z)/2,np.max(Z)))
     plt.figure()
     plt.bar(theta,Z/2)
     plt.xlim(-3, 3)
     plt.ylim(0, 100)
     plt.xlabel('bias as theta')
     plt.ylabel('Correct classifications [%]')
     plt.show()
```

```
At 20 degree and theta of 0.360000 the p is maximum of 71.000000%
for 142 correct classifications!
```



(d)

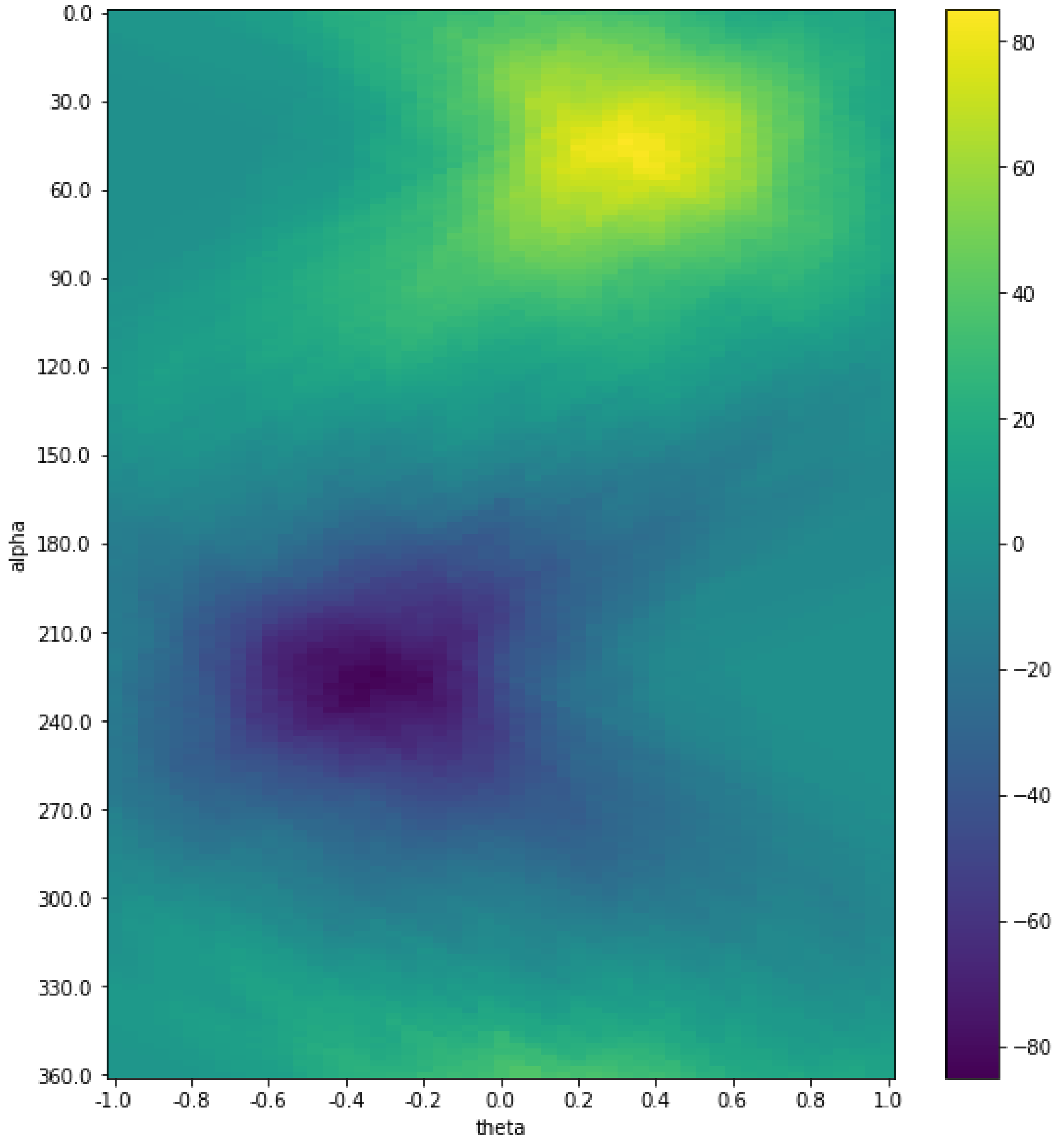
```
[6]: aNewData = np.genfromtxt('applesOranges.csv', delimiter = ',')[1:]
     theta = 0.36
     for j in range(np.shape(aData)[0]):
         aNewData[j,2] = SLCN(aData[j,0:2],W[3],theta)
     ix = np.arange(-1,1.5,0.1)
     itan = -(np.cos(a[3])/(np.sin(a[3])))
     ibias = - theta/np.sin(a[3])
     plt.figure()
     plt.scatter(aNewData[:,0],aNewData[:,1],c=(aNewData[:,2]))
     plt.quiver(*origin, np.cos(a[3]), np.sin(a[3]), scale=5, color = 'r')
     plt.plot(ix,ix*itan-ibias)
     plt.xlabel('X.1')
     plt.ylabel('X.2')
     plt.xlim(-2, 2)
     plt.ylim(-2, 2)
     plt.axis('equal')
     plt.show()
```



the weight vector (red vector) as feature detection vector. Any stimulous that falls on the side of blue line (feature seperation) where the vector is pointing is considered to be with calculated output of one! In another word, if the dot product of input and the weight vector is positiv, it falls into the yellow category (output one). every connectionist neuron can be interpreted as a hyperplane in a feature space, cutting the space into two parts. Data points are then projected onto weight vector.

(e)

```
[7]: aF = np.pi*np.linspace(0, 360, 181, endpoint=True)/180
WF = np.vstack(((np.cos(aF)), (np.sin(aF)))).T
thetaF = np.linspace(-1, 1, 51, endpoint=True)
YF = np.zeros((181,51))
for i in range(181):
    for j in range(51):
        for k in range(200):
            if SLCN(aData[k,0:2],WF[i],thetaF[j]) == aData[k,2]:
                YF[i,j] += 0.5
            else:
                YF[i,j] -= 0.5
plt.figure(figsize=(9,10))
heatmap = plt.imshow(YF)
plt.xlabel('theta')
plt.ylabel('alpha')
plt.xticks(np.linspace(0,50,11),np.linspace(-1,1,11))
plt.yticks(np.linspace(0,180,13),np.linspace(0,360,13))
plt.axis('auto')
plt.colorbar(heatmap)
plt.show()
iMax = int((np.argmax(YF))/51)
jMax = ((np.argmax(YF))%(iMax*51))
print('max performance of %f%% at theta = %f and alpha = %d degree'
      %(np.max(YF), thetaF[jMax], np.round(180*aF[iMax]/np.pi)))
```



```
max performance of 85.000000% at theta = 0.320000 and alpha = 44 degree
```

(f)

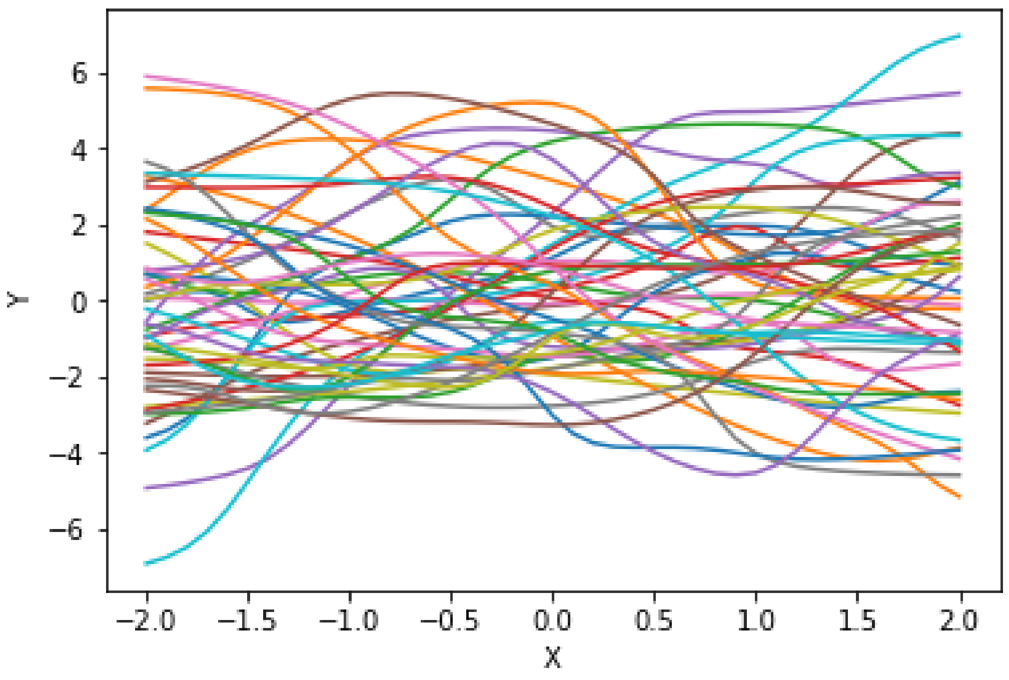
It depends on the size of the possible inputs and outputs and the dimentions of input and output. This method of optimization will be limited to the computation of power and time. But can be used in sensible ranges and percision. One of the places that this would prove inapplicable is in hand written letter recognition, the number of inputs can be 10*10 pixels which is 100 inputs {0,1} and 52 possible outputs {A,a,B,b,C,c…,Z,z}, the number of possiblities equalls: $52^{2^{100}}$ and the number of weights to optimize is minimum 100+100+52 (for one hidden layer with 100 neurons) and 64+52 biases (impractical dimensions to go through)…

2.2: Multi-layer Perceptrons

[8]:
```
def MLP(W,A,X,B):
    return np.sum(np.matmul(W.T,np.tanh(A*(X-B))))
```

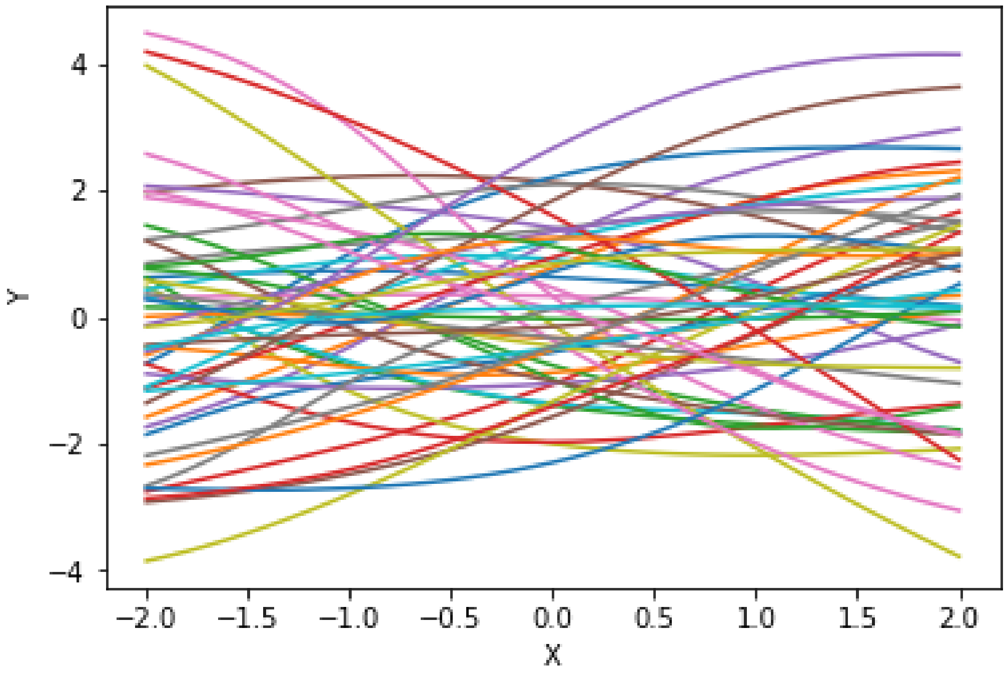
(a)

[9]:
```
X = np.linspace(-2, 2, 41, endpoint=True)
A0 = np.random.normal(0,2,(50,10)).reshape(50,10,1)
W = np.random.normal(0,1,(50,10)).reshape(50,10)
B = np.random.uniform(-2,2,(50,10)).reshape(50,10,1)
plt.figure()
Y0 = np.zeros((50,41))
for j in range(50):
    Y0[j] = (np.matmul(W[j],np.tanh(A0*(X-B))[j]))
plt.plot(X,Y0.T)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



(b)

```
[10]: A1 = np.random.normal(0,0.5,(50,10)).reshape(50,10,1)
      plt.figure()
      Y1 = np.zeros((50,41))
      for j in range(50):
          Y1[j] = (np.matmul(W[j],np.tanh(A1*(X-B))[j]))
      plt.plot(X,Y1.T)
      plt.xlabel('X')
      plt.ylabel('Y')
      plt.show()
```



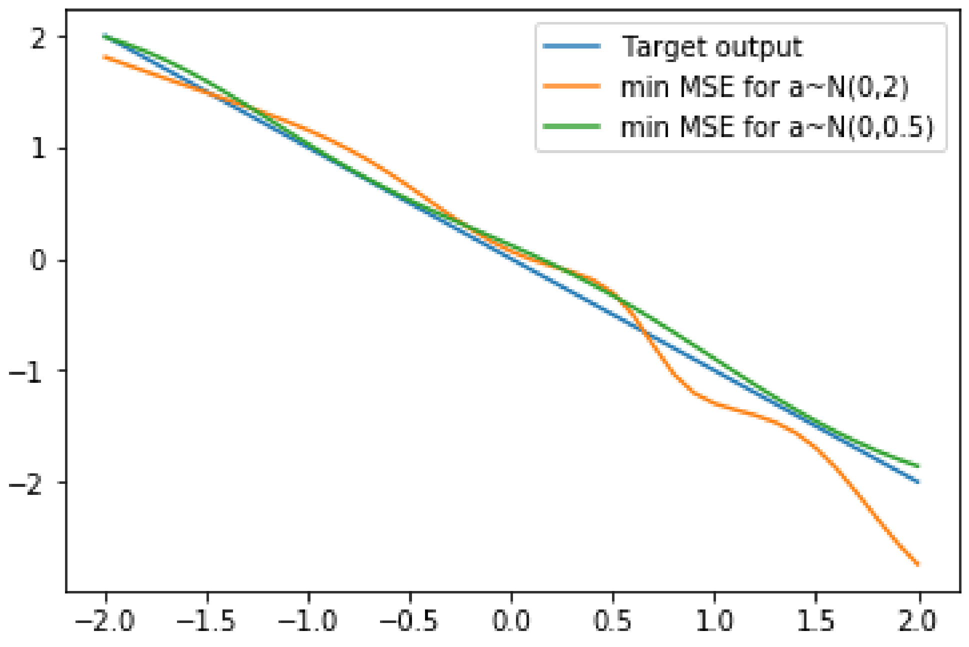
as can be seen in the 2nd plot, those with smaller variances show more linear behaviour (lines seem more straight) comparing to the first set of neurons with more complex behaviour. One may say that in the second set of neurons, the nonlinearity is supressed (limited by the smaller range of the variances).

(c)

```
[11]: def _MSE_(a,b):
          return (np.square(a-b).mean(axis=1))
```

```
[12]: MSE0 = _MSE_(-X,Y0)
      MSE1 = _MSE_(-X,Y1)
      print('For a~N(0,2) the best weight vector is:\n\t%s'
            %W[np.argmin(MSE0)])
      print('\n')
      print('For a~N(0,0.5) the best weight vector is:\n\t%s'
            %W[np.argmin(MSE1)])
      plt.figure()
      plt.plot(X,-X)
      plt.plot(X,Y0[np.argmin(MSE0)])
      plt.plot(X,Y1[np.argmin(MSE1)])
      plt.legend(('Target output','min MSE for a~N(0,2)',
                  'min MSE for a~N(0,0.5)'))
      plt.show()
```

```
For a~N(0,2) the best weight vector is:
        [-0.21665853  1.02431059 -1.78778343 -0.09802248  0.69102602 -0.29471823
 -0.54956206 -0.96740441 -1.03980558  0.67983907]


For a~N(0,0.5) the best weight vector is:
        [-1.67458058 -1.29555813 -2.17270713 -0.25517644 -1.1247645   0.81814111
 -0.19212582 -1.22858975  0.81591698 -0.08953906]
```


Target output
min MSE for a~N(0,2)
min MSE for a~N(0,0.5)