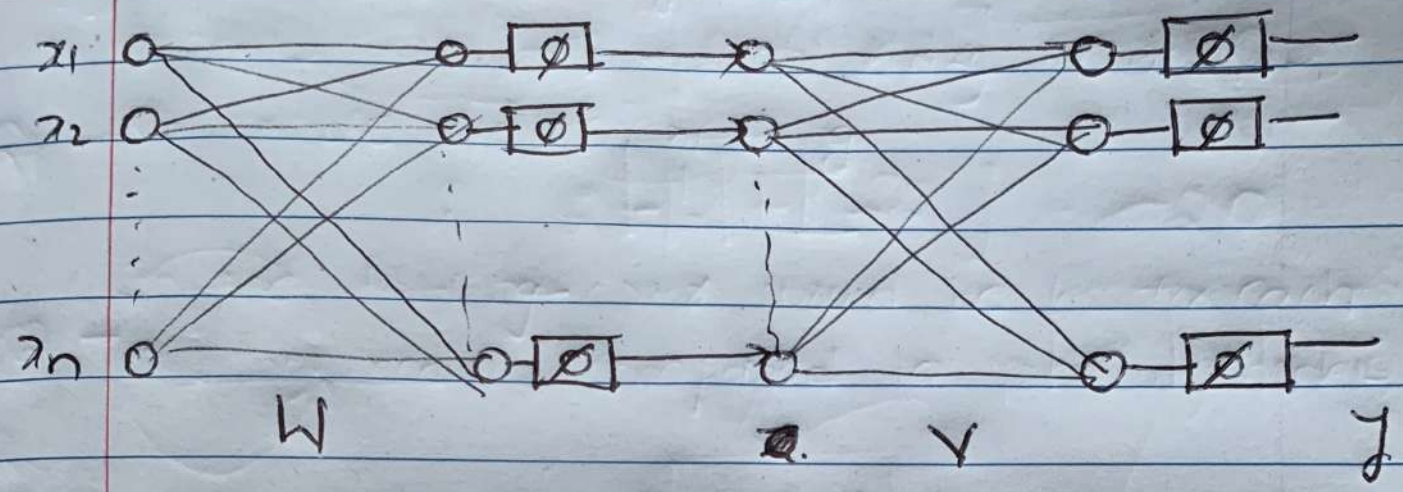
Name: Kalyan Kumar Paladugula

UIN: 679025059.

Neural Networks Midterm Sp2020

Q1) Given Network



$$y = \phi(V\{\phi(WX)\}) \quad ; \quad E = 1/2\,\|d-y\|^2.$$

$\eta = 1$

$$z = (d-y) \odot \phi'(V\phi(WX))$$

$$\frac{\partial E}{\partial V} = -\frac{1}{2} \cdot 2\,(d-y) \odot \phi'(V\phi(WX)) \cdot \phi(WX)$$

$$= -z \cdot \phi(WX)$$

$$\frac{\partial E}{\partial W} = \left[\left[-\frac{1}{2} \cdot 2 \cdot (d-y) \odot \phi'(V\phi(WX) \cdot V \cdot\right] \odot \phi'(WX)\right] \cdot X$$

$$= -\left[(z \cdot V) \odot \phi'(W \cdot X)\right] \cdot X.$$

$$V \leftarrow V + z \cdot \phi(WX)$$

$$W \leftarrow W + \left[(z \cdot V) \odot \phi'(W \cdot X)\right] \cdot X.$$

2) Single layer FF
n-neurons
Zero biases

$w_1, w_2 - - w_n$ are neuron weight vectors
$||w_1|| = ||w_2|| = - - - ||w_n||$

Let $x_1, x_1 - - x_m$ are input nodes.
X be the vector(input)

Consider the euclidean distance b/w input vector X and a weight vector w_i

$$d_i = \sqrt{||X - w_i||^2} = \sqrt{(X - w_i)^T \cdot (X - w_i)}$$

$$= \sqrt{X^T X - X^T w_i - w_i^T X + w_i^T w_i}$$

$$= \sqrt{||X||^2 + ||w_i||^2 - 2 X^T \cdot w_i}$$

Let $V_i = X^T \cdot w_i$ - Local field of neuron

$$d_i = \sqrt{||X||^2 + ||w_i||^2 - 2V_i}$$

since $||X||^2$ and $||w_i||^2$ are constant for all neuron's

For neuron 'i' to be the winner neuron V_i should be maximum

since $d_i = \sqrt{||X||^2 + ||w_i||^2 - 2V_i}$

$\Downarrow\Downarrow\Downarrow$

d_i should be minimum

In other words, winner neuron is the one whose weight vector is closest to an input in terms of Euclidean distance

(Q3) one input,
no bias
one output with step activation fn.

initial weight = 0

a) $w_{i+1} \leftarrow w_i + xy$

Initial weight $w_0 = 0$

i) First input $y_0 = \text{step}(w_0 \cdot x_0) = \text{step}(0) = 1$

New weight $w_1 = 1$

ii) second input $y_1 = \text{step}(w_1 \cdot x_1) = 1$

$w_2 = 1 + 1 \cdot 1 = 2.$

$w_3 = 3$

$w_4 = 4$

Limit of $w = \infty$

b) $w_1 \leftarrow \alpha w_0 + x_0 y_0$

$w_1 = x_0 y_0$

First Input $y_0 = \text{step}(w_0 \cdot x_0) = 1$

$w_1 = 1 \cdot 1 = 1$

Second Input $x_1 = 1$

$y_1 = \text{step}(w_1 \cdot x_1) = 1$

$w_2 = \alpha + 1$

$w_3 = 1 + \alpha + \alpha^2$

$1 + \alpha + \alpha^2 + \alpha^3 + \cdots + \infty$

Geometric progression

as $0 < \alpha < 1$, Limit of $W = \frac{1}{1-\alpha}$.

c) $w \leftarrow \alpha w + xy$.

$0 < \alpha < 1$

$w_0 = 0$

training sequence: $0, 1, 0, 1, 0, 1 \ldots$

First Input:

$x_0 = 0 \quad w_0 = 0$

$y_0 = \text{step}(0 \cdot 0) = 1$

$w_1 = \alpha \cdot 0 + 0 \cdot 1 = 0$

Second input:

$x_1 = 1, \quad w_1 = 0$

$y_1 = 1$

$w_2 = \alpha \cdot w_1 + 1 \cdot 1$

$= 1$

Third input:

$x_2 = 0, \quad w_2 = 1$

$y_2 = 1$

$w_3 = \alpha \cdot w_2 + 0$

$= \alpha$

Fourth Input:

$x_3 = 1, \quad w_3 = \alpha.$

$y_3 = \text{step}(\alpha \cdot 1) = 1$

$w_4 = \alpha^2 + 1.$

Fifth Input

$x_4 = 0, \quad w_4 = \alpha^2 + 1.$

$y_4 = 1$

$w_5 = \alpha(\alpha^2 + 1) + 0$

$= \alpha^3 + \alpha.$

Sixth Input:

$x_5 = 1, \quad w_5 = \alpha^3 + \alpha.$

$y_5 = 1$

$w_6 = \alpha(\alpha^3 + \alpha) + 1.$

$= \alpha 1 + \alpha^2 + \alpha^4$

w_1 odd 1.

w_2 even α.

w_3 odd α^2+1

w_4 even $\alpha^3+\alpha$.

$\alpha^4+\alpha^2+1$.

$\alpha^5+\alpha^3+\alpha$.

~~$1+\alpha$ α^2+1 $\alpha(\alpha^2+1)$~~

i) <u>when n is even:</u>

$$\alpha+\alpha^3+\alpha^5 \ldots = \alpha(1+\alpha^2+\alpha^4+\ldots)$$

common ratio $= \alpha^2$

Geometric progression

as $0<\alpha<1$ Limit of $w = \frac{\alpha}{1-\alpha^2}$.

ii) <u>when n is odd</u>

$1+\alpha^2+\alpha^4 \ldots$

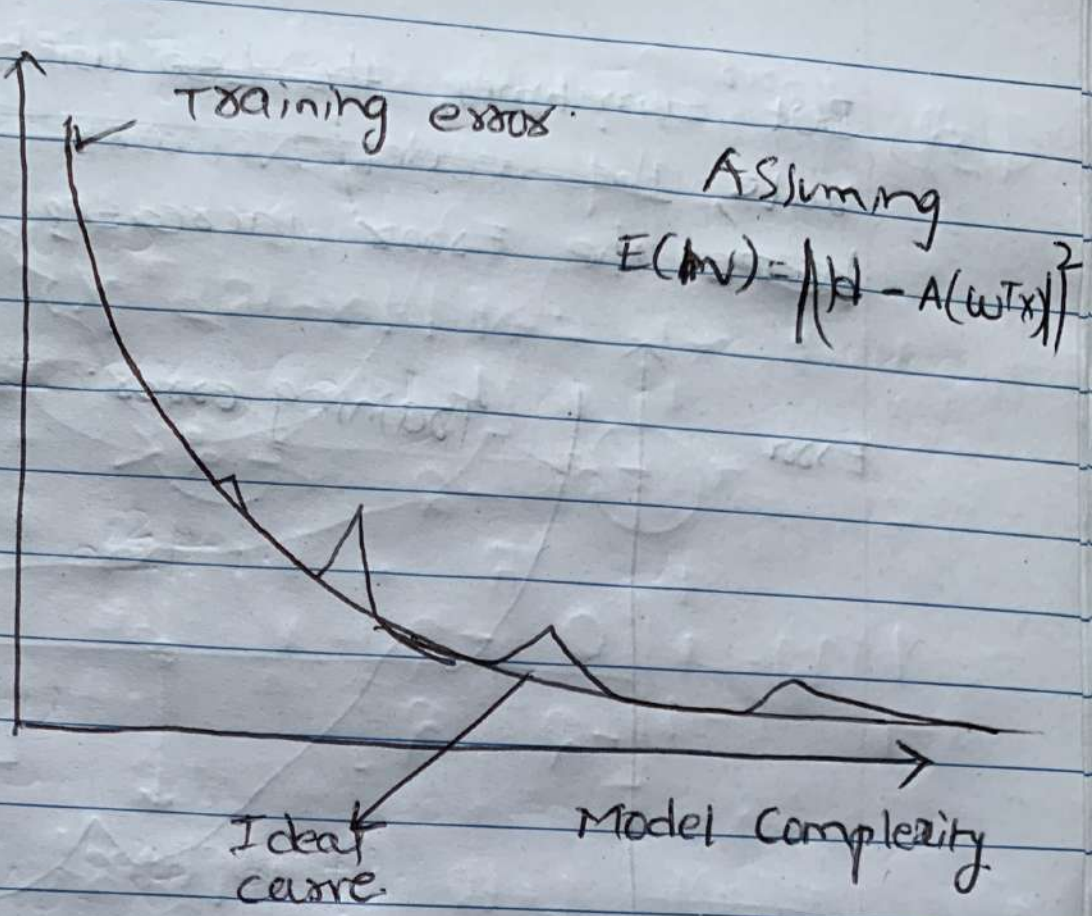
common ratio $= \alpha^2$

<u>G.P</u>

as $0<\alpha<1$ $\lim_{w\to\infty} = \frac{1}{1-\alpha^2}$.

4)

a)

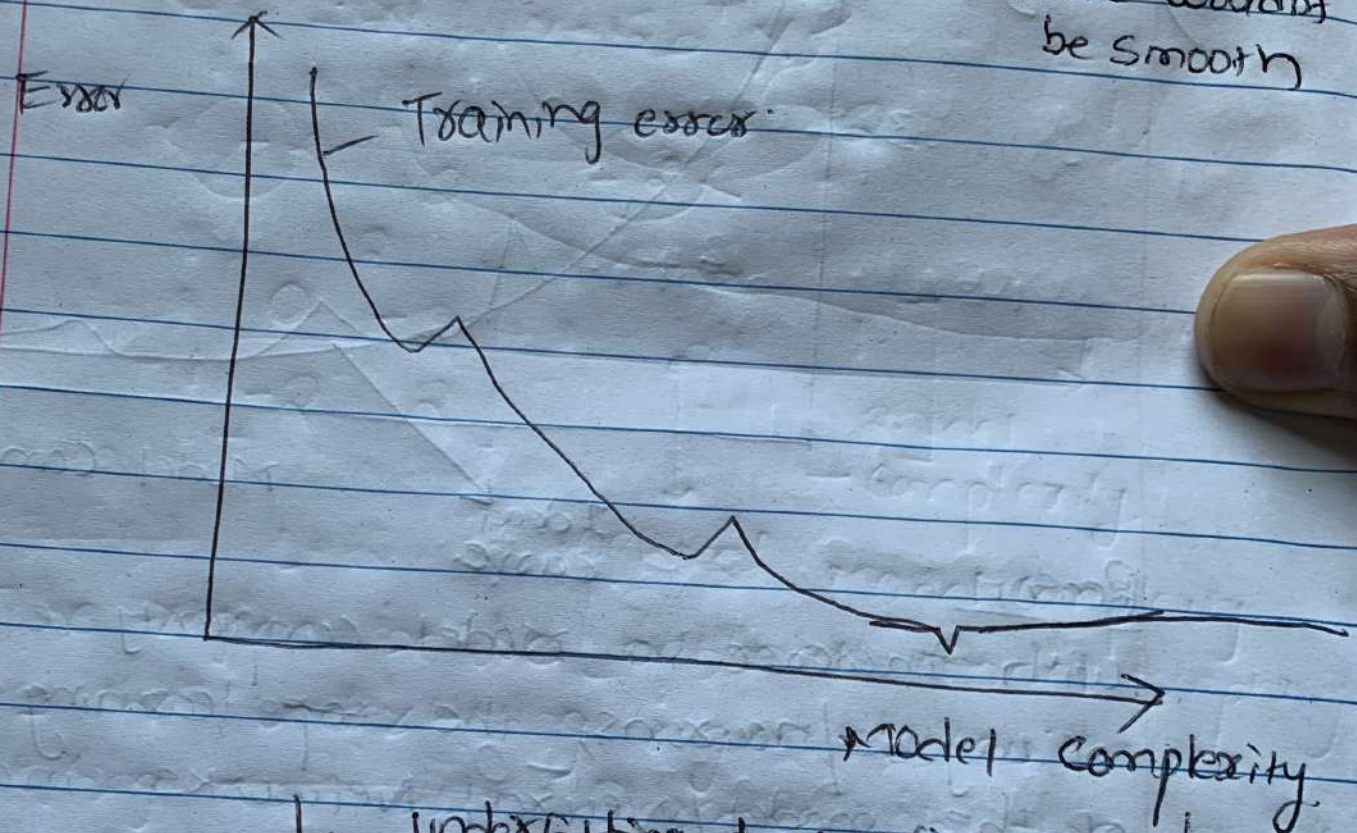


Reason:

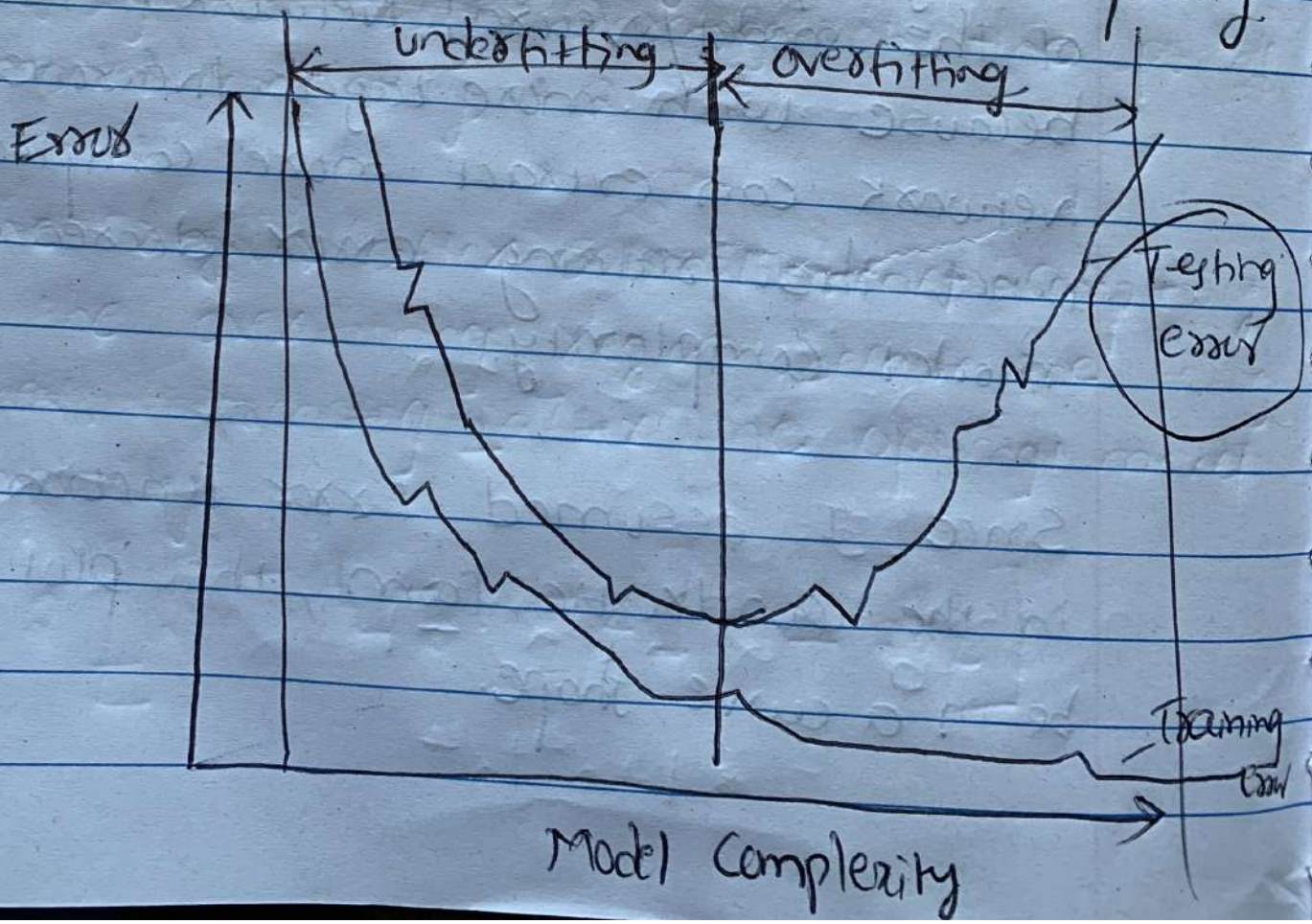
With Increase in Model complexity ie number of Layers/neurons, the ~~cap~~ learning capacity of the model/Neural Networks increases because with more Layers/Neurons, the Network can extract more complex features Hence, the training error decreases with model complexity.

Since I assumed Error function to be in the Quadratic form, the plot would be in a curve shape.

But Sometimes, ~~the loss~~ with high learning rate, ~~the model/Network skips the minima~~ ~~here~~ the Error increases and curve would not be smooth

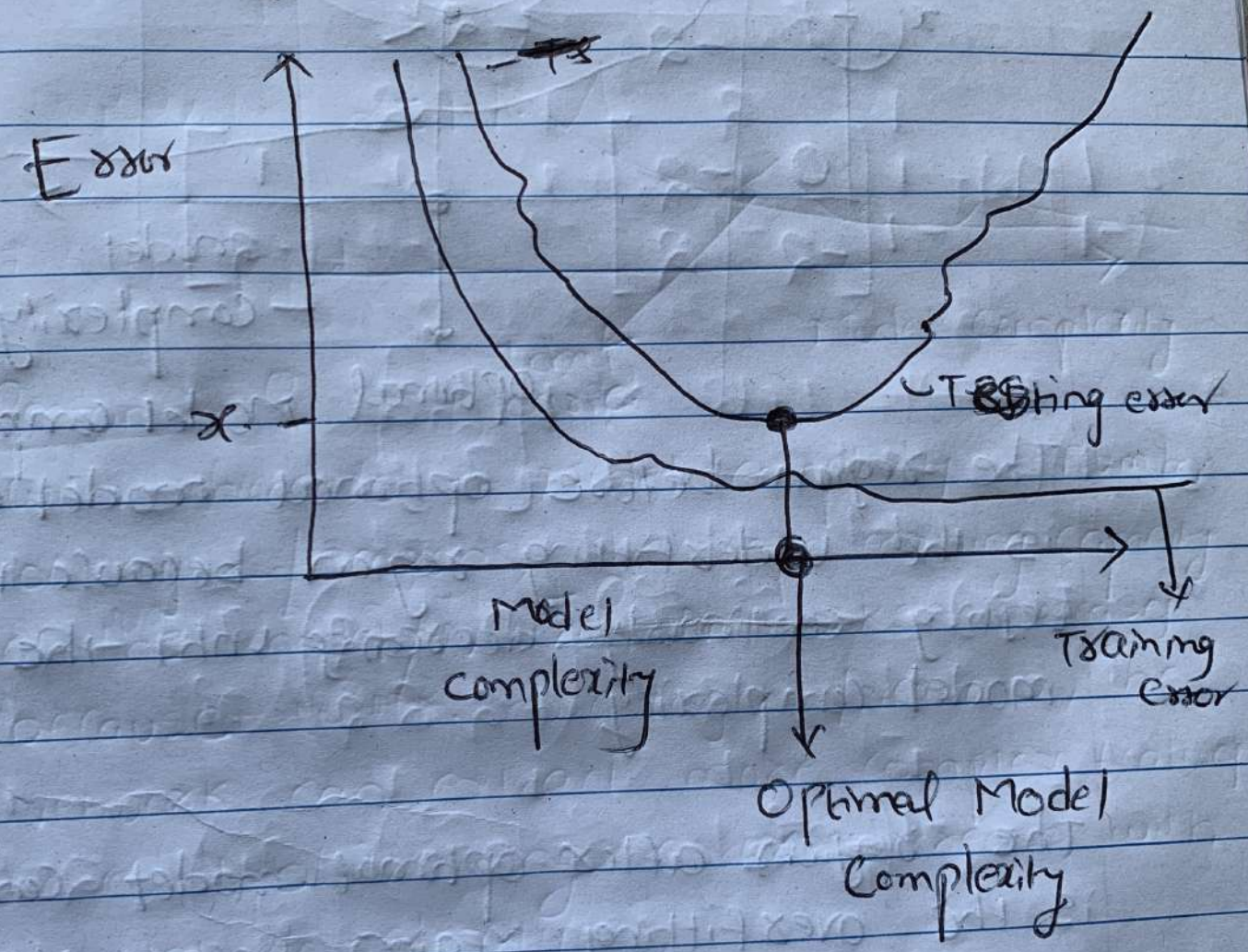


b)



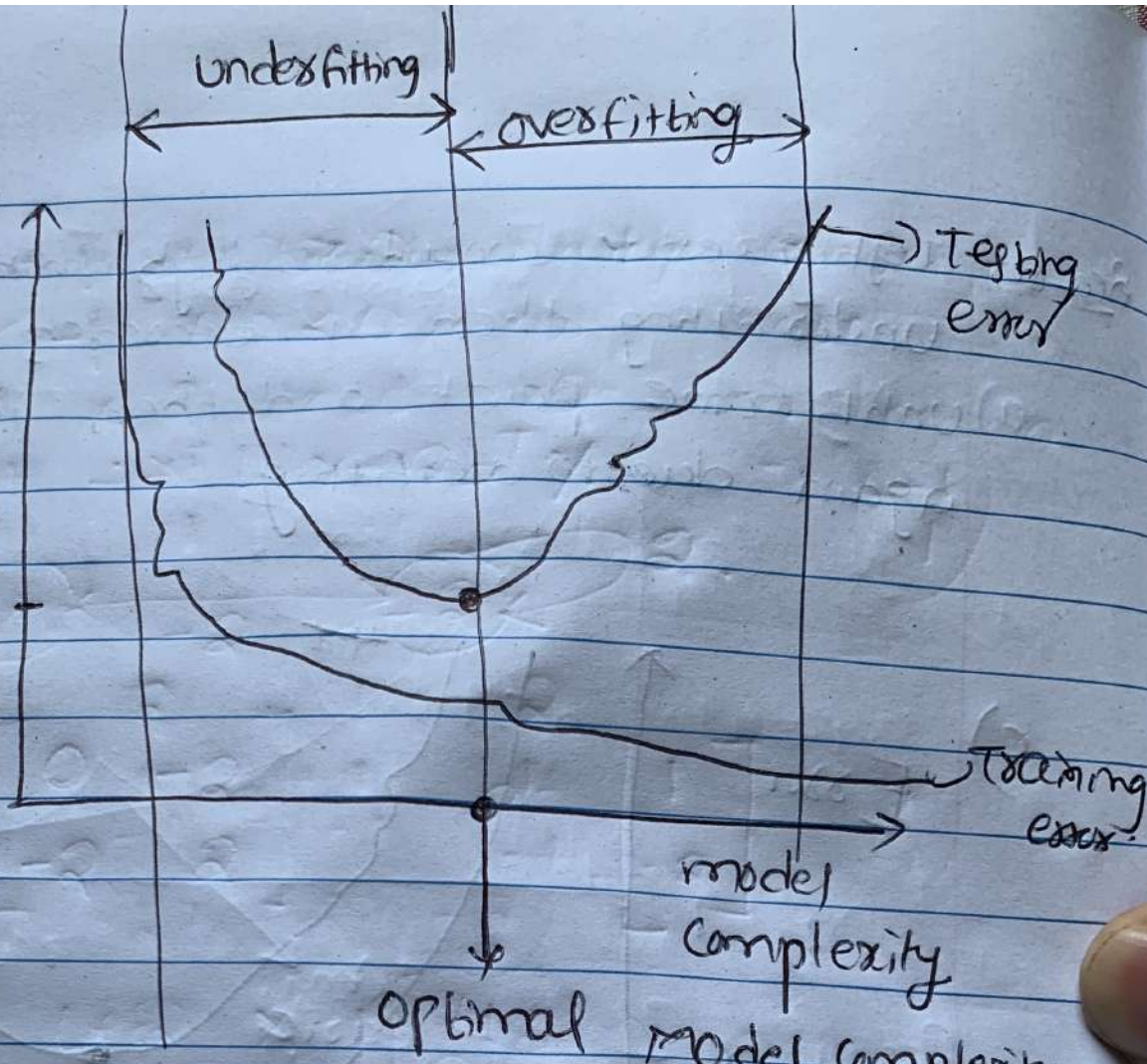
Reason: Testing error ~~would~~ is high initially due to underfitting then it decreases gradually until some point and then it increases again due to overfitting

c)



Our model should perform well on unseen data i.e testing data. Hence, optimal model complexity would be the one where testing error is minimum

d)



The region before optimal model complexity is the underfitting region because the testing ~~continuously~~ decreases until the optimal model complexity

The region after optimal model complexity is the overfitting region because the testing starts increasing after the optimal model complexity

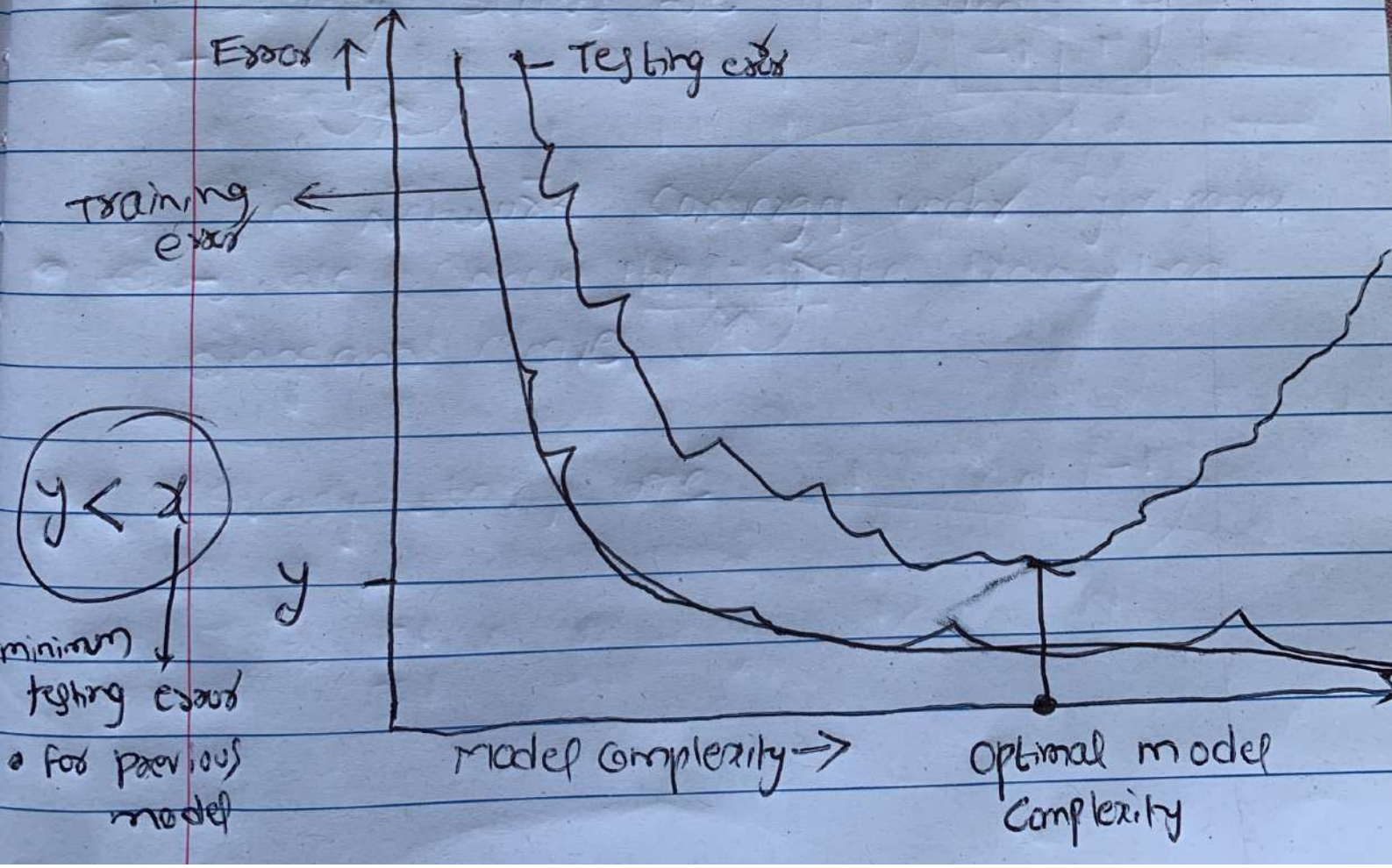
4)

e) With more training data, the variances of the models decreases which make them less vulnerable to overfitting.

So, the testing error would be less for each of the models compared to previous models.

Hence, even the high complex model would not overfit.

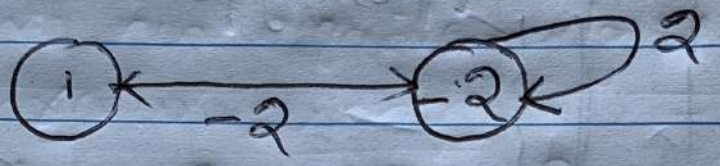
→ we would get broad error (testing curve).



Sometimes, ~~we would~~ the optimal model complexity increases due to decrease in variance and testing error.

(Q5) $\phi(x) = 1$ if $x \geq 0$
-1 if $x < 0$

$E(x) = \frac{1}{2} x^T W x - b^T x$



$$W = \begin{bmatrix} 0 & -2 \\ -2 & 2 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

~~$E(x)$~~ Let $x = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix}$

$$E(x) = -(\tilde{x}_1 \ \tilde{x}_2)\begin{pmatrix} 0 & -2 \\ -2 & 2 \end{pmatrix}\begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} - 2(1 \ -2)\begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix}$$

$$= (-2\tilde{x}_2 \quad 2\tilde{x}_1 - 2\tilde{x}_2)\begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} - 2(\tilde{x}_1 - 2\tilde{x}_2)$$

$$= 2\left[\tilde{x}_1\tilde{x}_2 + (\tilde{x}_2)^2 - \tilde{x}_1\tilde{x}_2\right]$$

$$= 2\left[2\tilde{x}_1\tilde{x}_2 - 2(\tilde{x}_2)^2 - \tilde{x}_1 + 2\tilde{x}_2\right]$$

$$E(x) = 2\left(2\tilde{x}_1\tilde{x}_1 - 2(\tilde{x}_2)^2 - \tilde{x}_1 + 2\tilde{x}_2\right)$$

a) $x_1 = \begin{pmatrix}1\\1\end{pmatrix}$, $x_2 = \begin{pmatrix}1\\-1\end{pmatrix}$, $x_3 = \begin{pmatrix}-1\\1\end{pmatrix}$, $x_4 = \begin{pmatrix}-1\\-1\end{pmatrix}$

$y_1 = \phi(-2\tilde{x}_2 + 1)$, $y_2 = \phi(-2\tilde{x}_1 + 2\tilde{x}_2 - 2)$

Asynchronous update:

i) updating second neuron first

$x_1 \Rightarrow \begin{pmatrix}1\\1\end{pmatrix} \Rightarrow \begin{matrix}2 & 1 & 2 & 1\\ 1 & 1 & 1 & 1\\ -1 & -1 & -1 & -1\end{matrix} = \begin{pmatrix}1\\-1\end{pmatrix}$

$x_2 \Rightarrow \begin{pmatrix}1\\-1\end{pmatrix} \Rightarrow \begin{matrix}2 & 1 & 2 & 1\\ 1 & 1 & 1 & 1\\ -1 & -1 & -1 & -1\end{matrix} = \begin{pmatrix}1\\-1\end{pmatrix}$

$x_3 \Rightarrow \begin{pmatrix}-1\\1\end{pmatrix} \Rightarrow \begin{matrix}2 & 1 & 2 & 1\\ -1 & -1 & -1 & -1\\ 1 & 1 & 1 & 1\end{matrix} = \begin{pmatrix}-1\\1\end{pmatrix}$

$x_4 \Rightarrow \begin{pmatrix}-1\\-1\end{pmatrix} \Rightarrow \begin{matrix}2 & 1 & 2 & 1\\ -1 & 1 & 1 & 1\\ -1 & -1 & -1 & -1\end{matrix} = \begin{pmatrix}1\\-1\end{pmatrix}$

ii) updating First Neuron First

$$x_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \Rightarrow \begin{matrix} 1 & 2 & 1 & 2 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & 1 \end{matrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$$x_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \Rightarrow \begin{matrix} 1 & 2 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{matrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$x_3 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \Rightarrow \begin{matrix} 1 & 2 & 1 & 2 \\ -1 & -1 & -1 & -1 \\ 1 & +1 & 1 & 1 \end{matrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$$x_4 = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \Rightarrow \begin{matrix} 1 & 2 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{matrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$E(x) = 2\left[2\tilde{x}_1\tilde{x}_2 - 2(\tilde{x}_2)^2 - \tilde{x}_1 + 2\tilde{x}_2\right]$$

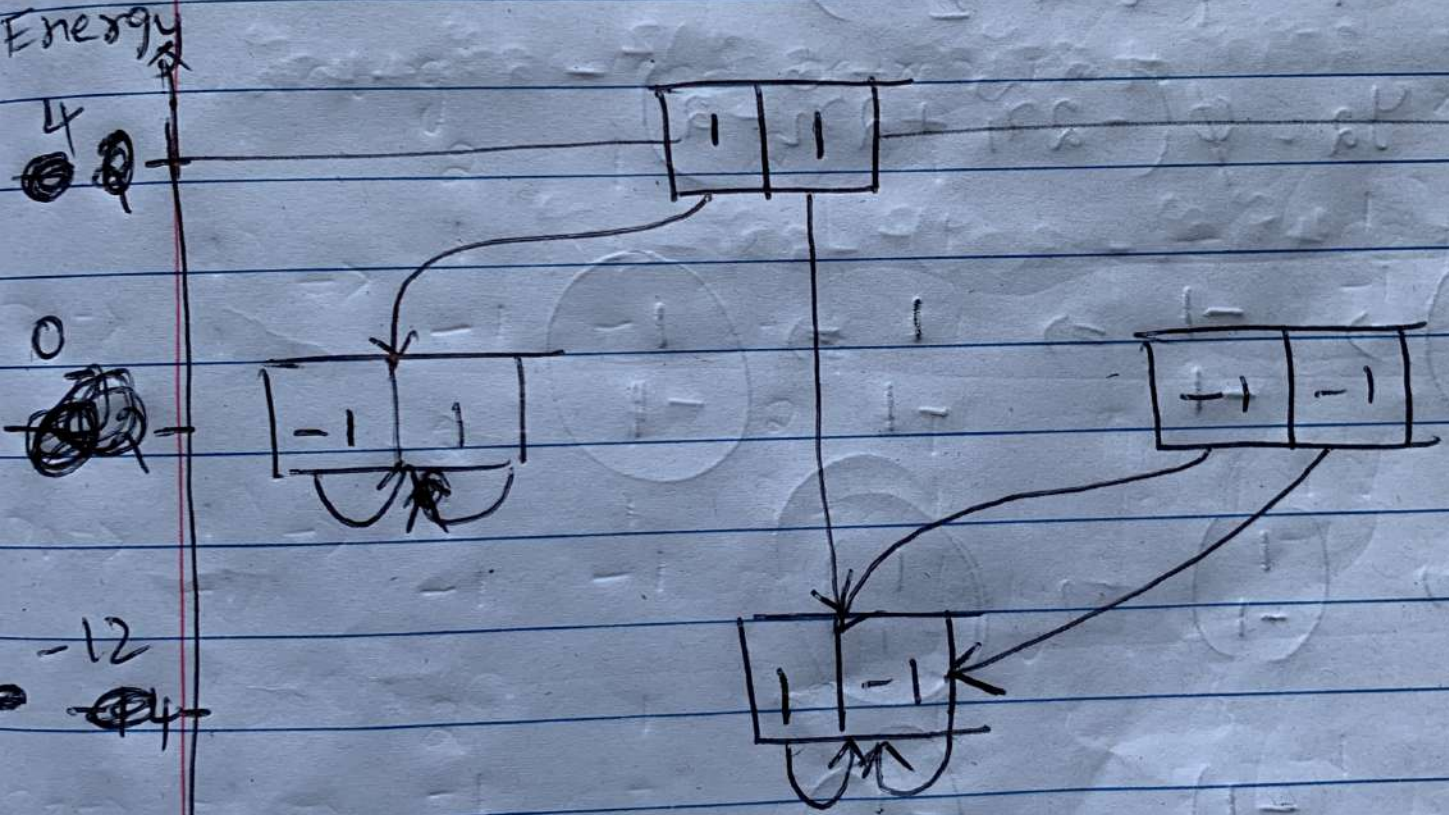
$$E(x_1) = 2\left[2 - 2 - 1 + 2\right] = 4$$

$E(x_2) = 2[-2 - 1 - 1 - 2] = -12$

$E(x_3) = 2[-2 - 1 + 1 + 2] = 0$

$E(x_4) = 2[2 - 1 + 1 - 2] = 0$

State transition diagram with Energy levels



Steady states are $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and $\begin{pmatrix} -1 \\ 1 \end{pmatrix}$

b) Yes. The Network always converge under the asynchronous update. Check the state transition diagram and also since the weight matrix is symmetric and has non-negative diagonal elements

c) synchronous update

$$y_1 = \phi(-2\tilde{x}_2 + 1)$$

$$y_2 = \phi(-2\tilde{x}_1 + 2\tilde{x}_2 - 2)$$

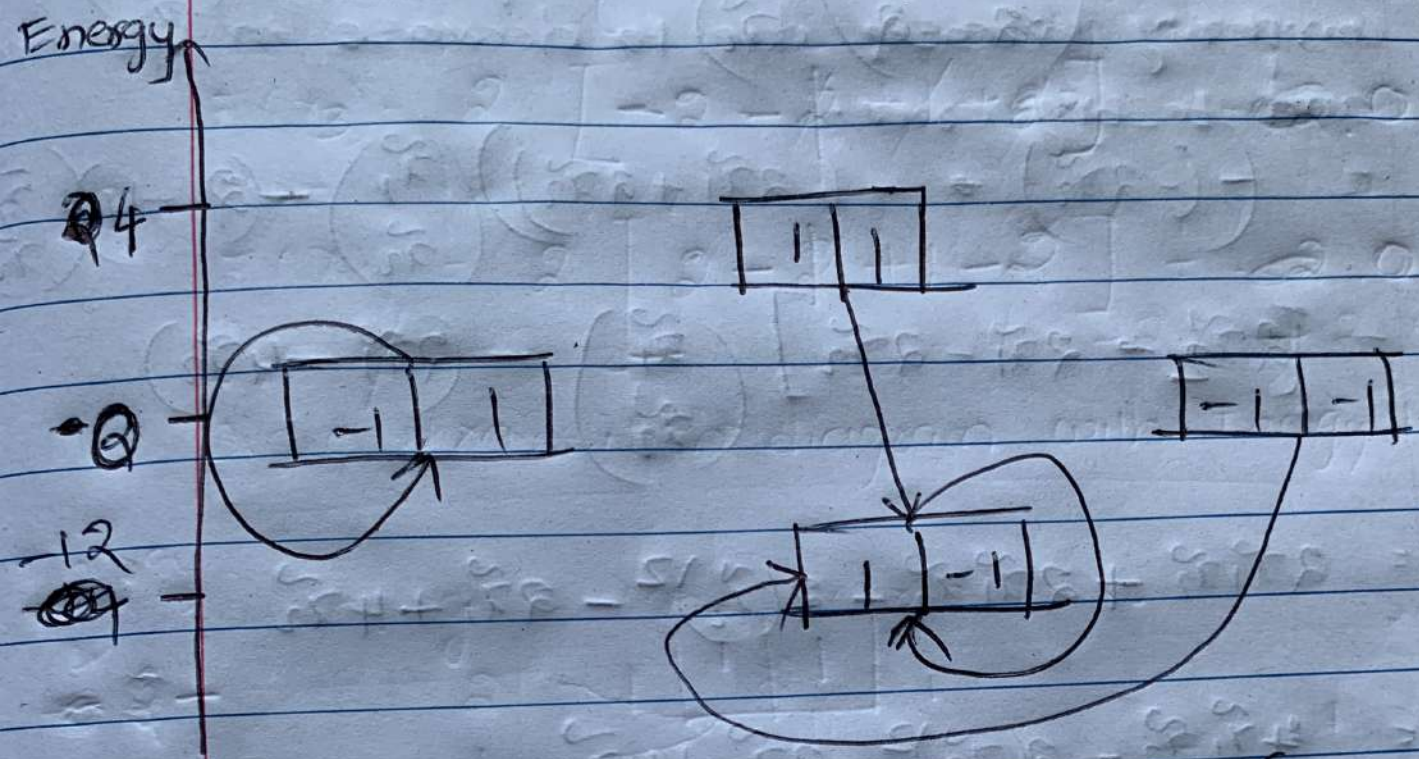
$x_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \Rightarrow \begin{matrix} -1 \\ -1 \end{matrix} \Rightarrow \begin{matrix} 1 \\ -1 \end{matrix} \Rightarrow \begin{matrix} 1 \\ -1 \end{matrix}$

$x_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 \\ -1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

$x_3 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} -1 \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

$x_4 = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 \\ -1 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

State transition diagram



Steady states are $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and $\begin{pmatrix} -1 \\ 1 \end{pmatrix}$

The network converges under Synchronous update. Check the state transition diagram above

In [47]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as mp
np.random.seed(100)
```

Using cvxopt library to solve the optimization problem of SVM

In [48]:
```
from cvxopt import matrix, solvers
```

In [49]:
```
x = np.random.uniform(low =0, high = 1, size=(100,2))
```

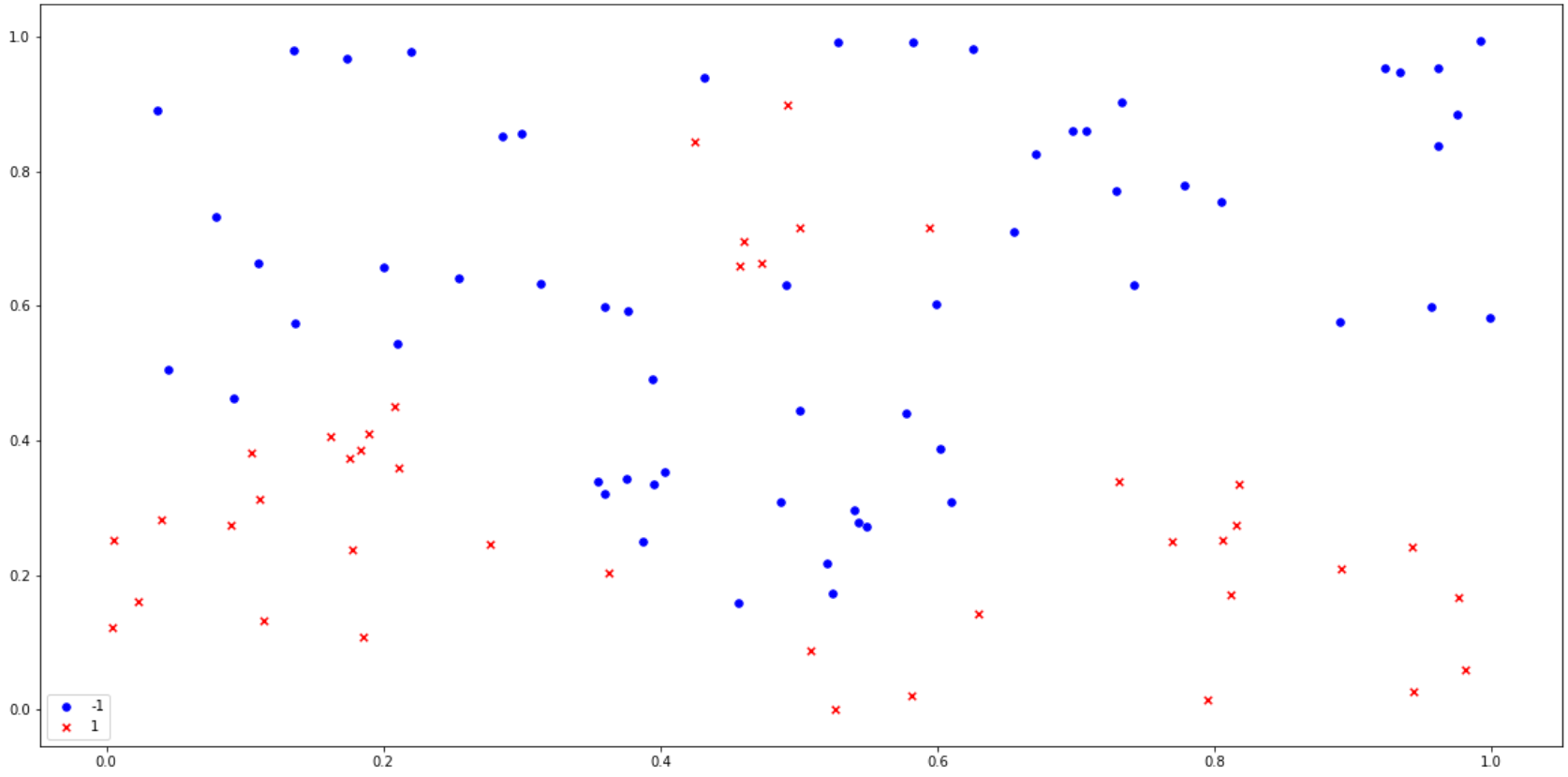
In [50]:
```
d =[]
colors =[]
for i in range(len(x)):
    if (x[i][1] < 0.2*(np.sin(10*x[i][0])) + 0.3) or ((x[i][1] - 0.8)**2 + (x[i][0]-0.5)**2 < 0.0225):
        d.append(1)
        colors.append("r")
    else:
        d.append(-1)
        colors.append("b")
d = np.asarray(d)
```

In [51]:
```
d.reshape(1,-1).shape
```

Out[51]:
```
(1, 100)
```

In [52]:
```
fig, ax = mp.subplots(figsize = (20,10))
cdict = {1: 'red', -1: 'blue'}
markers = {1:'x', -1:'o'}
for g in np.unique(d):
    ix = np.where(d == g)
    ax.scatter(x[:,0][ix], x[:,1][ix], c = cdict[g],marker = markers[g],label = g, s = 30)

ax.legend(loc = 'lower left')
```

Out[52]: <matplotlib.legend.Legend at 0x23f13959d68>



Kernel Function

I am using a polynomial kernel of degree 3

In [62]:
```
def kernel(X, Y):
#     K = np.zeros((X.shape[0], Y.shape[0]))

#     for i, x in enumerate(X):
#         for j, y in enumerate(Y):
#             K[i, j] = np.exp(-0.5*np.linalg.norm(x - y) ** 2)

    return (1+X.dot(Y.T))**3
```

Finding Alpha values using cvxopt library to solve the optimization problem of SVM

In [63]:
```
def train_svm():
    n, k = x.shape
    y_matrix = d.reshape(1, -1) * 1.
    H = np.dot(y_matrix.T, y_matrix) * kernel(x, x)
    P = matrix(H)
    q = matrix(-np.ones((n, 1)))
    G = matrix(np.diag(np.ones(n) * -1))
    h = matrix(np.zeros(n))
    A = matrix(y_matrix)
    b = matrix(np.zeros(1))

#     solvers.options['abstol'] = 1e-10
#     solvers.options['reltol'] = 1e-10
#     solvers.options['feastol'] = 1e-10

    return solvers.qp(P, q, G, h, A, b)
```

In [64]:
```
parameters = train_svm()
```

```
     pcost       dcost       gap    pres   dres
 0: -6.4862e+01 -1.9051e+02  4e+02  1e+01  3e+00
 1: -2.3482e+02 -3.8691e+02  2e+02  6e+00  2e+00
 2: -3.1394e+02 -4.9169e+02  2e+02  5e+00  1e+00
 3: -5.3124e+02 -7.5929e+02  3e+02  5e+00  1e+00
 4: -1.4735e+03 -1.7590e+03  3e+02  4e+00  1e+00
 5: -2.1726e+03 -2.5225e+03  4e+02  4e+00  1e+00
 6: -6.5390e+03 -7.2515e+03  7e+02  4e+00  1e+00
 7: -2.8736e+04 -3.0840e+04  2e+03  4e+00  1e+00
 8: -5.0928e+04 -5.4486e+04  4e+03  4e+00  1e+00
 9: -1.1426e+05 -1.2310e+05  9e+03  4e+00  1e+00
10: -2.5077e+05 -2.7840e+05  3e+04  4e+00  1e+00
11: -4.4253e+05 -5.0925e+05  7e+04  3e+00  1e+00
12: -7.9750e+05 -9.6717e+05  2e+05  3e+00  8e-01
13: -1.2592e+06 -1.5351e+06  3e+05  1e+00  4e-01
14: -1.3478e+06 -1.3794e+06  3e+04  9e-02  2e-02
15: -1.3483e+06 -1.3486e+06  3e+02  9e-04  2e-04
16: -1.3483e+06 -1.3483e+06  3e+00  9e-06  2e-06
17: -1.3483e+06 -1.3483e+06  3e-02  9e-08  2e-08
Optimal solution found.
```

Finding Bias using the alpha values

In [65]:
```
def bias(alphas):

# Threshold to find the support vectors
# Instead of zero tolerance, I am using some floating point tolerance

    threshold = 1e-5
    sv = (alphas > threshold).reshape(-1, )
    b = []
    for i in range(len(d[sv])):
        sum = 0
        for j in range(len(x)):
#             print(x[sv][i])
#             print(kernel(x[j].reshape(1,2), x[sv][i].reshape(1,2)))
#             print(f)
            sum += alphas[j]*d[j]*kernel(x[j].reshape(1,2), x[sv][i].reshape(1,2))
#             print(sum)
        b.append(d[sv][i] - sum)

#     print(b)
#     print(f)
    b = np.mean(b)
#     print(b)
    return b, sv
```

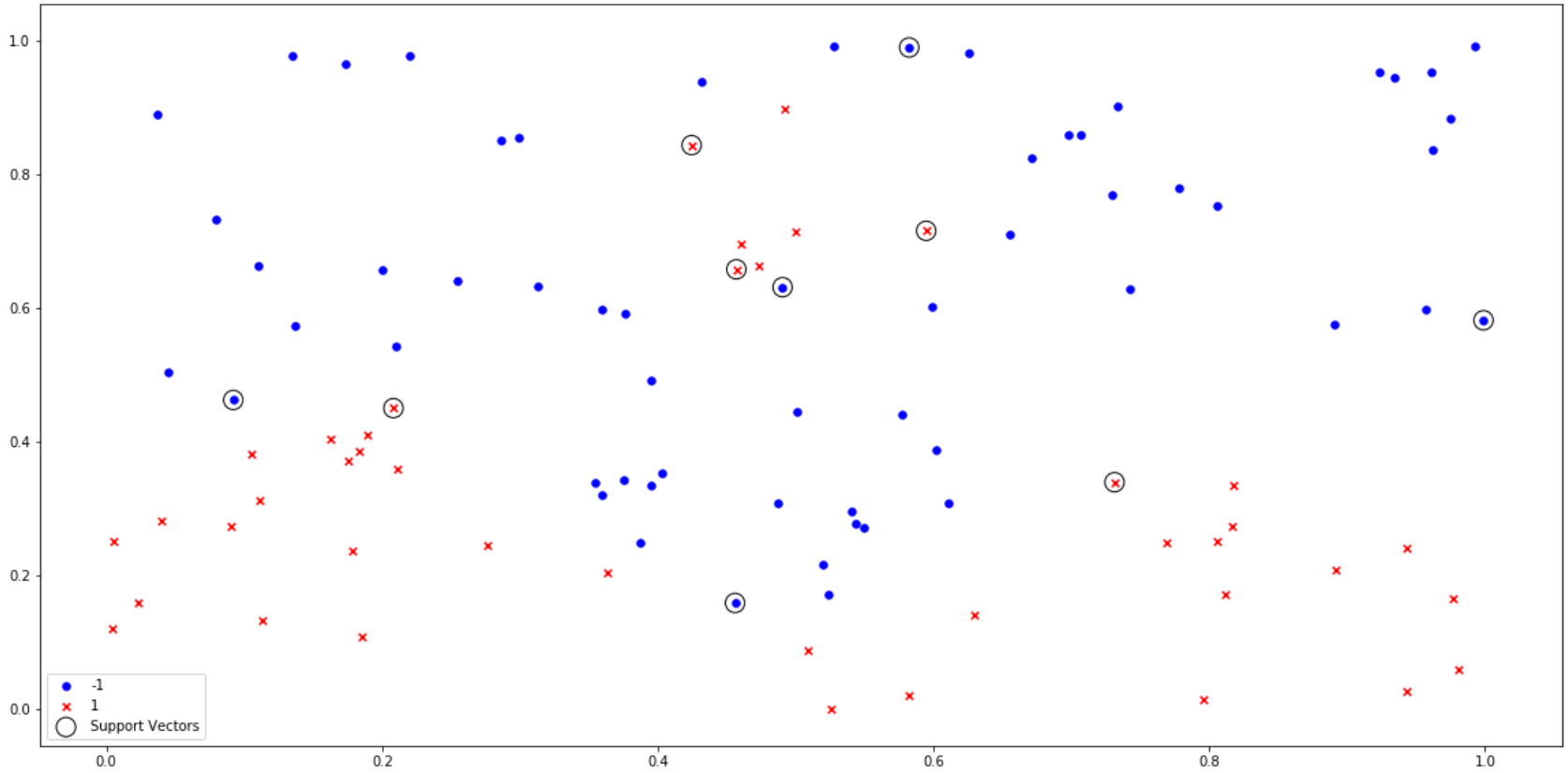
In [68]:
```
alphas = np.array(parameters['x'])[:, 0]
print(len(alphas))
b, sv = bias(alphas)

print('Alpha values of suppport vectors:', alphas[sv])
print('bias = ', b)
```

```
100
Alpha values of suppport vectors: [141624.83651654  97252.79769487 219453.89958003 114956.89032222
 105263.05297173  44391.42974999 551375.68098238 443111.91242398
  57475.12259771 921738.6130395 ]
bias =  228.77513944812785
```

Plotting support vectors

In [73]:
```
fig, ax = mp.subplots(figsize = (20,10))
cdict = {1: 'red', -1: 'blue'}
markers = {1:'x', -1:'o'}
for g in np.unique(d):
    ix = np.where(d == g)
    ax.scatter(x[:,0][ix], x[:,1][ix], c = cdict[g],marker = markers[g],label = g, s = 30)

support_vectors = x[sv]
ax.scatter(support_vectors[:,0], support_vectors[:,1], s=200, facecolors='none', edgecolors='black', label =
"Support Vectors")
ax.legend(loc='lower left')
```

Out[73]: <matplotlib.legend.Legend at 0x23f142d9da0>

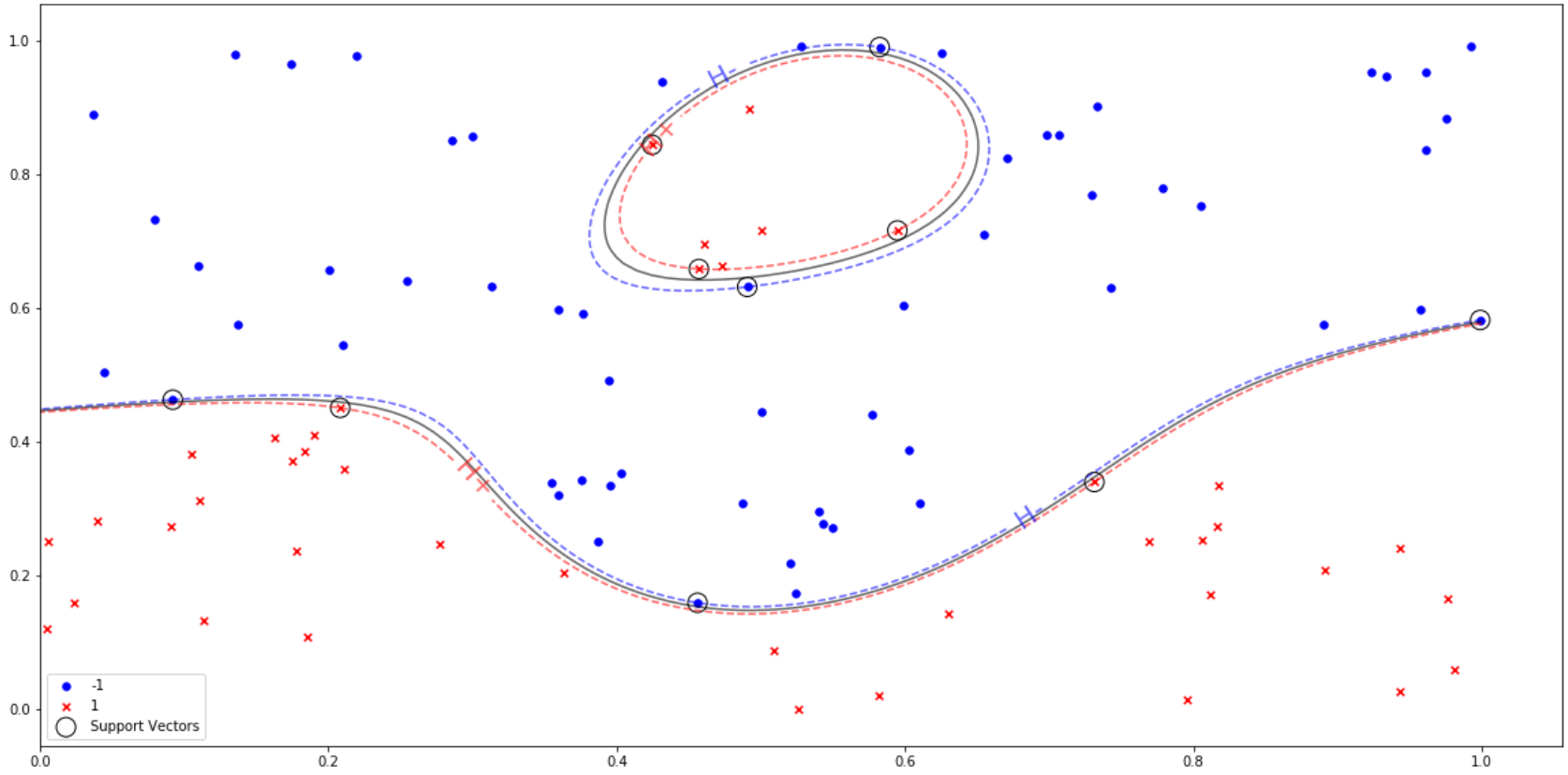


Finding the three hyperplanes

In [74]:
```
def decision_function(x, y):
    p =[]
    for i in x:
        for j in y:
            p.append(kernel(i,j))
    return np.array(p).reshape(x.shape[0], y.shape[0])
```

In [76]:
```
xx =  np.linspace(0,1,100)
yy = np.linspace(0,1,100)
XX, YY = np.meshgrid(xx, yy)
# print(XX.shape)
xy = np.c_[XX.ravel(), YY.ravel()]
#print(xy)

df = decision_function(x,xy)
#print(((alphas*d).reshape(1,XX.shape[0])).dot(kxy))
Z = ((((alphas*d).reshape(1,XX.shape[0])).dot(df)) +b).reshape(XX.shape)

fig, ax = mp.subplots(figsize = (20,10))
contour  = ax.contour(XX, YY, Z, colors=['blue', 'black', 'red'], levels = [-1,0,1], alpha = 0.6, linestyles
= ['--','-','--'])

fmt = {}
strs = ['H-', 'H', 'H+']
for l, s in zip(contour.levels, strs):
    fmt[l] = s

# Label every other level using strings
ax.clabel(contour, contour.levels[::2], inline = True,inline_spacing = 10, fmt=fmt, fontsize=20)



cdict = {1: 'red', -1: 'blue'}
markers = {1:'x', -1:'o'}
for g in np.unique(d):
    ix = np.where(d == g)
    ax.scatter(x[:,0][ix], x[:,1][ix], c = cdict[g],marker = markers[g],label = g, s = 30)

# ax.scatter(x[:,0], x[:,1], c= colors, s=10, label = ('blue'))
# ax.scatter(x[:,0], x[:,1], c= colors, s=10, label = ('red'))

# ax.legend()
support_vectors = x[sv]
ax.scatter(support_vectors[:,0], support_vectors[:,1], s=200, facecolors='none', edgecolors='black', label =
"Support Vectors")
ax.legend(loc='lower left')
```

Out[76]: <matplotlib.legend.Legend at 0x23f14971ba8>



In [45]:
```
# def af(x):
#     if x>= 0:
#         return 1
#     else:
#         return -1
```

In [ ]:
```
# def predict(x,y):

#     f =[]
#     for k in range(len(x)):
#         p =[]
#         for i in range(len(x[0])):
#             q = np.asarray([x[0][i], y[0][i]])
#             sum = 0
#             for j in range(len(x[0])):
#                 r = np.asarray([x[0][j], y[0][j]])
#                 sum += alphas[j] * d[j] * kernel(r.reshape(1,2), q.reshape(1,2))
#             p.append(af(sum+b))
#         f.append(p)

#     return np.asarray(f).T
```

In [ ]:
```
# u = []
# v = []
# for i in range(len(x)):
#     if predict(b, x[i]) == 1:
#         u.append(x[i])
#     elif predict(b,x[i]) == -1:
#         v.append(x[i])
# u = np.asarray(u)
# v = np.asarray(v)
```

In [119]:
```
# import numpy as np
# import matplotlib.pyplot as plt
# from sklearn import svm
# from sklearn.datasets import make_blobs


# # we create 40 separable points
# X, y = make_blobs(n_samples=40, centers=2, random_state=6)

# # fit the model, don't regularize for illustration purposes
# clf = svm.SVC(kernel='linear', C=1000)
# clf.fit(X, y)

# plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# # plot the decision function
# ax = plt.gca()
# xlim = ax.get_xlim()
# ylim = ax.get_ylim()

# # create grid to evaluate model
# xx = np.linspace(xlim[0], xlim[1], 30)
# yy = np.linspace(ylim[0], ylim[1], 30)
# YY, XX = np.meshgrid(yy, xx)
# # print(xx)
# # print(XX)
# # print(o)
# xy = np.vstack([XX.ravel(), YY.ravel()]).T
# Z = clf.decision_function(xy).reshape(XX.shape)
# # print(Z)
# # print(f)

# # plot decision boundary and margins
# ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
#            linestyles=['--', '-', '--'])
# # plot support vectors
# ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
#            linewidth=1, facecolors='none', edgecolors='k')
# plt.show()
```

In [ ]:

In [ ]: