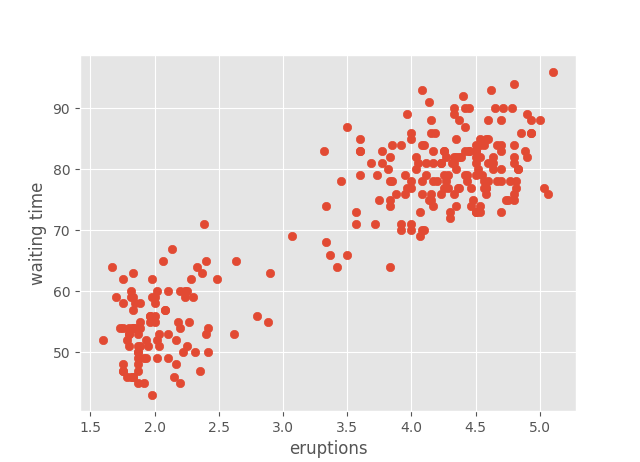
**ID5030 :: Quiz-2**

Varun V (AE14B050)

**Solution 1: Clustering Old Faithful Geyser Data:**

**(a) Clustering Models:**

There are a number of algorithms at our disposal when we want to cluster data. Each of them works well with a specific kind of clustering job while are marginally worse at other kinds. Let us first plot our current data to determine the kind of clustering algorithm we may use for best results.



From a direct look at the data we see the number of cluster are most probably 2 and they have a well-defined boundary albeit a few outlying cases.

We shall use the Scikit-learn library functions for quick and efficient implementation of the clustering algorithms.

The most basic clustering algorithms we use are as follows:

* Density Based Clustering: This method assumes the clusters are high density areas while the spare points are generally noise or outliers which it ignores. DBSCAN is a popular density based clustering algorithm.
* Distribution Based Clustering: Popular probability distribution functions are fit to the data and boundaries are defined using the parameters of the fit functions. This work really well when there exists and underlying distribution, sometimes almost perfectly clustering data. EM algorithm of Gaussian mixtures is generally used.
* Centroid Based Clustering: Here we try to find central vectors which cluster data. These algorithms require the number of clusters we are trying to find, a drawback, but are fast compared to many other algorithms. They also tend to find equal sized clusters which means they perform very poorly on density or distribution based clustering. K-means algorithm and its variants are used.

**(b)Simple Clustering Program:**

**Ref:** [**http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN**](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN)**.**

From our observations we use the density based spatial clustering algorithm modified for noise. We obtain a clear clustered output with some of the data points as outliers (in violet).



**Code:**

1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Sat Apr 14 14:07:30 2018
5. @author: Varun
7. Q1 :  clustering
8. """
9. **import** numpy as np
10. **import** pandas as pd
11. **import** matplotlib.pyplot as plt
12. #import tensorflow as tf
13. **from** sklearn.model\_selection **import** train\_test\_split
14. **from** sklearn.preprocessing **import** StandardScaler,MinMaxScaler
15. **from** sklearn.cluster **import** DBSCAN,AgglomerativeClustering,KMeans
16. #from sklearn.mixture import BayesianGaussianMixture,GaussianMixture
17. **from** matplotlib **import** style
18. style.use('ggplot')
20. """
21. Importing the data and standard scaling it
22. """
23. data = pd.read\_csv('oldfaithful.csv')
24. scaler = StandardScaler()
25. data.iloc[:,:] = scaler.fit\_transform(data.iloc[:,:])
27. """
28. plot for intial observations
29. """
30. #plt.xlabel(data.columns[0])
31. #plt.ylabel(data.columns[1]+' time')
32. #plt.scatter(data.iloc[:,0],data.iloc[:,1])
34. """
35. DBSCAN clustering algorithm implementation
36. """
37. clusterer = DBSCAN(eps = 0.4,min\_samples = 10)#use eps = 0.52 for no outliers
38. db = clusterer.fit(data)
39. labels = db.labels\_ #labels for the data points we fit the algorithm to
41. """
42. plot
43. """
44. plt.xlabel(data.columns[0])
45. plt.ylabel(data.columns[1]+' time')
46. scatterplot = plt.scatter(data.iloc[:,0],data.iloc[:,1], c = labels)

**(c) Naïve Bayes Classifier:**

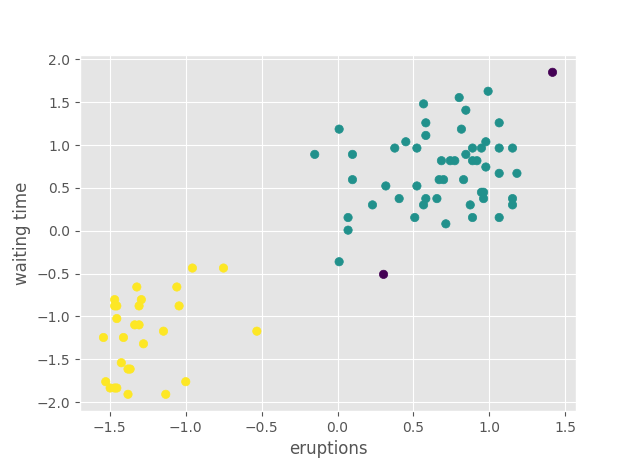
**Code:**

1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Sat Apr 14 14:07:30 2018
5. @author: Varun
7. Q1 :  clustering
8. """
9. **import** numpy as np
10. **import** pandas as pd
11. **import** matplotlib.pyplot as plt
12. #import tensorflow as tf
13. **from** sklearn.model\_selection **import** train\_test\_split
14. **from** sklearn.preprocessing **import** StandardScaler,MinMaxScaler
15. **from** sklearn.cluster **import** DBSCAN,AgglomerativeClustering,KMeans
16. #from sklearn.mixture import BayesianGaussianMixture,GaussianMixture
17. **from** sklearn.naive\_bayes **import** GaussianNB
18. **from** matplotlib **import** style
19. style.use('ggplot')
21. """
22. Importing the data and standard scaling it
23. """
24. data = pd.read\_csv('oldfaithful.csv')
25. scaler = StandardScaler()
26. data.iloc[:,:] = scaler.fit\_transform(data.iloc[:,:])
28. """
29. plot for intial observations
30. """
31. #plt.xlabel(data.columns[0])
32. #plt.ylabel(data.columns[1]+' time')
33. #plt.scatter(data.iloc[:,0],data.iloc[:,1])
35. """
36. DBSCAN clustering algorithm implementation
37. """
38. clusterer = DBSCAN(eps = 0.4,min\_samples = 10)#use eps = 0.52 for no outliers
39. db = clusterer.fit(data)
40. labels = db.labels\_ #labels for the data points we fit the algorithm to
42. """
43. plot
44. """
45. #plt.xlabel(data.columns[0])
46. #plt.ylabel(data.columns[1]+' time')
47. #scatterplot = plt.scatter(data.iloc[:,0],data.iloc[:,1], c = labels)
48. """
49. Gaussian Naive bayes classifier - as data is continuous
50. """
51. Xtrain,Xtest,ytrain,ytest = train\_test\_split(data,labels,test\_size = 0.3)
52. classifier = GaussianNB()
53. classifier.fit(Xtrain,ytrain)
54. ypred = classifier.predict(Xtest)
55. accuracy = classifier.score(Xtest,ytest)
57. plt.xlabel(Xtest.columns[0])
58. plt.ylabel(Xtest.columns[1]+' time')
59. scatterplot = plt.scatter(Xtest.iloc[:,0],Xtest.iloc[:,1], c = ypred)#prediction data
60. #scatterplot = plt.scatter(Xtest.iloc[:,0],Xtest.iloc[:,1], c = ytest) #true data

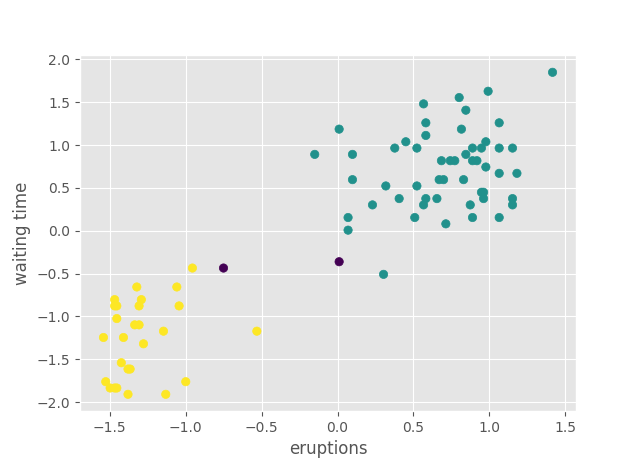
We use the Gaussian Naïve Bayes classifier as our data is continuous. Fractional and even negative. The Gaussian estimation handles these use cases for us.

Below are the plots for the 30% test data, both true and predicted. We do not neglect the outliers to fit our classifier. As observed the predictions are highly accurate with a prediction accuracy of 95.122%.

True labels:

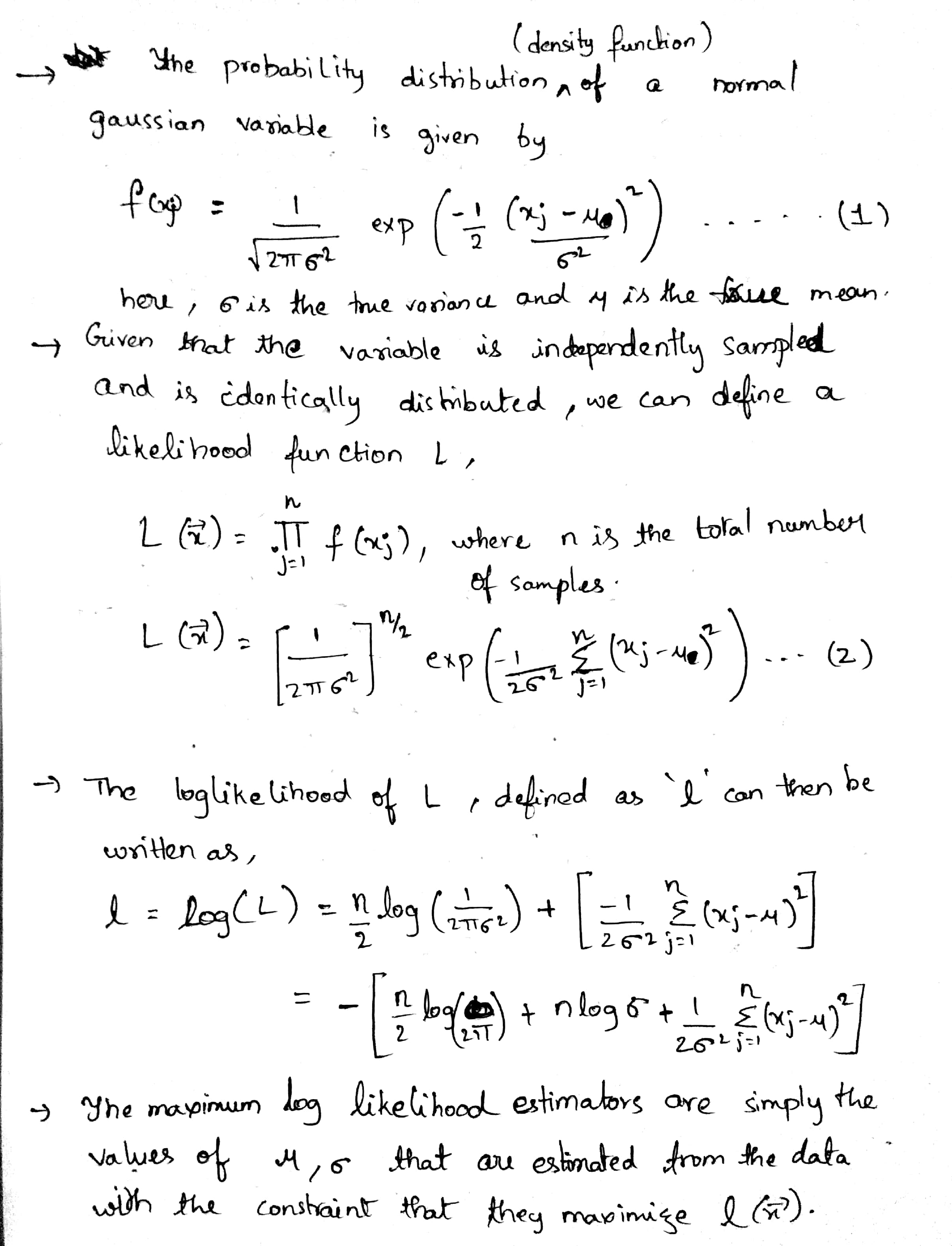


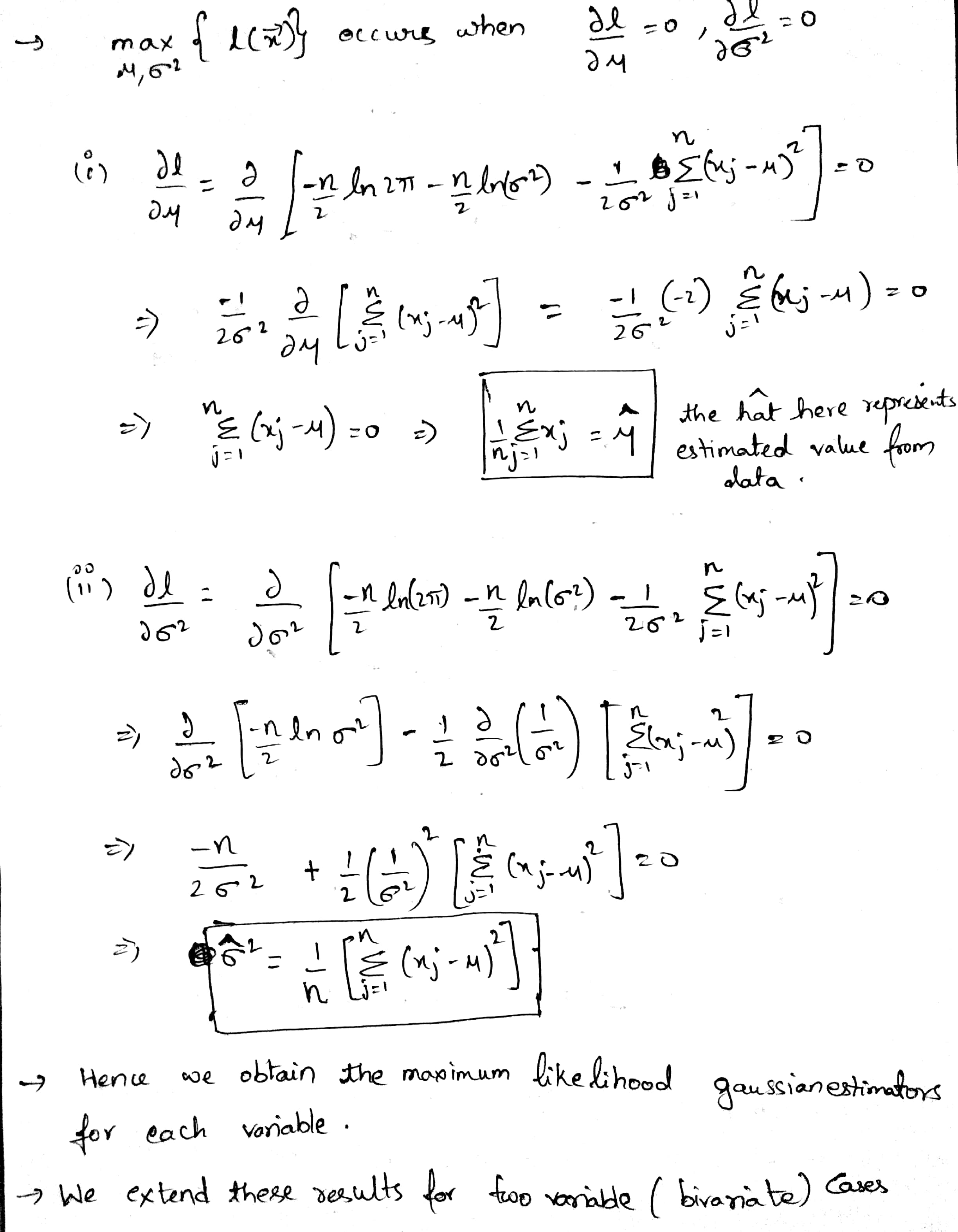
NB predicted labels:

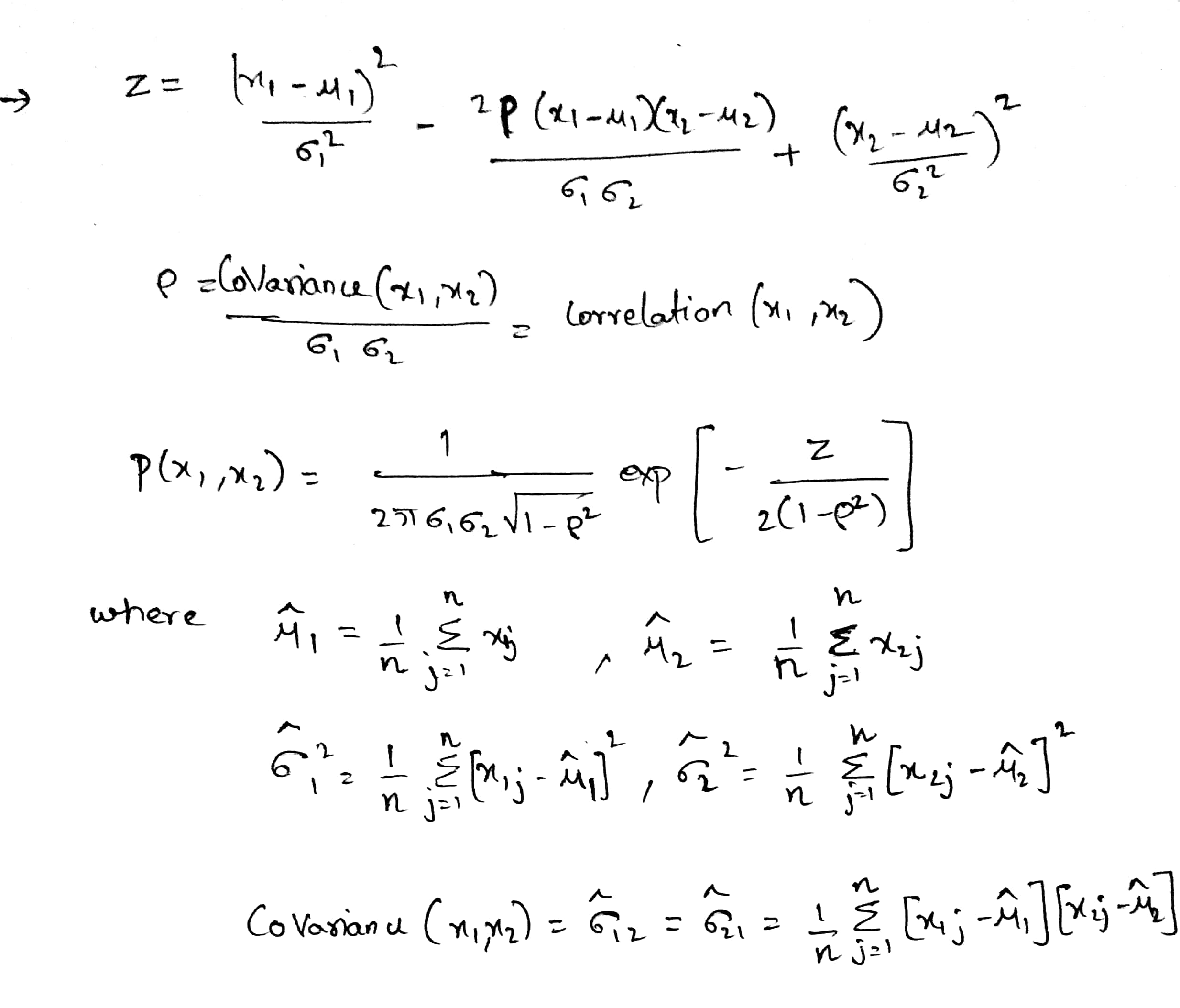


**(d) Maximum Likelihood Gaussians:**

Following a detailed derivation on Gaussian estimator parameters.







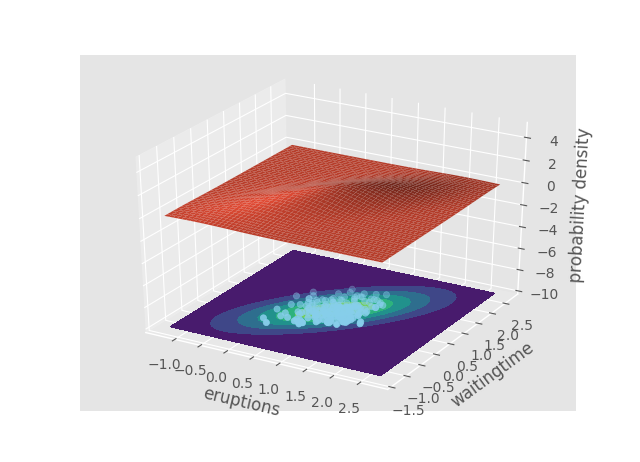
We use these above Bivariate relations in our code below to estimate our Gaussian distribution over the given data.

**Code:**

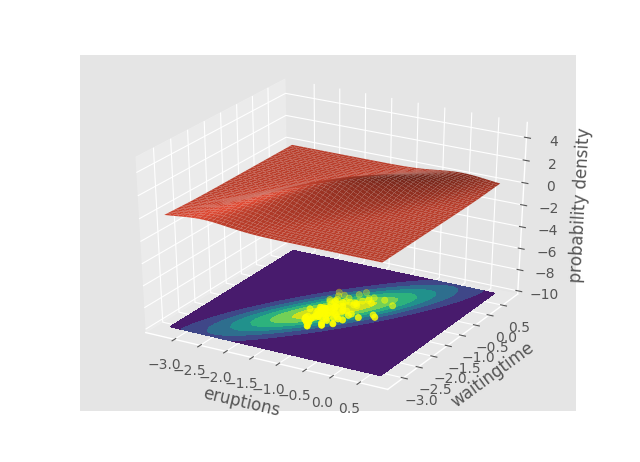
1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Sat Apr 14 14:07:30 2018
5. @author: Varun
7. Q1 :  clustering
8. """
9. **import** numpy as np
10. **import** pandas as pd
11. **import** matplotlib.pyplot as plt
12. #import tensorflow as tf
13. **from** sklearn.model\_selection **import** train\_test\_split
14. **from** sklearn.preprocessing **import** StandardScaler,MinMaxScaler
15. **from** sklearn.cluster **import** DBSCAN,AgglomerativeClustering,KMeans
16. #from sklearn.mixture import BayesianGaussianMixture,GaussianMixture
17. **from** sklearn.gaussian\_process **import** GaussianProcessRegressor
18. **from** sklearn.naive\_bayes **import** GaussianNB
19. **from** matplotlib **import** style
20. **from** matplotlib.patches **import** Ellipse
21. **from** matplotlib **import** cm
22. **from** mpl\_toolkits.mplot3d **import** Axes3D
23. style.use('ggplot')
25. """
26. Importing the data and standard scaling it
27. """
28. data = pd.read\_csv('oldfaithful.csv')
29. scaler = StandardScaler()
30. data.iloc[:,:] = scaler.fit\_transform(data.iloc[:,:])
32. """
33. plot for intial observations
34. """
35. #plt.xlabel(data.columns[0])
36. #plt.ylabel(data.columns[1]+' time')
37. #plt.scatter(data.iloc[:,0],data.iloc[:,1])
39. """
40. DBSCAN clustering algorithm implementation
41. """
42. clusterer = DBSCAN(eps = 0.4,min\_samples = 10)#use eps = 0.52 for no outliers
43. db = clusterer.fit(data)
44. labels = db.labels\_ #labels for the data points we fit the algorithm to
46. """
47. plot
48. """
49. #plt.xlabel(data.columns[0])
50. #plt.ylabel(data.columns[1]+' time')
51. #scatterplot = plt.scatter(data.iloc[:,0],data.iloc[:,1], c = labels)
52. """
53. Gaussian Naive bayes classifier - as data is continuous
54. """
55. Xtrain,Xtest,ytrain,ytest = train\_test\_split(data,labels,test\_size = 0.3)
56. classifier = GaussianNB()
57. classifier.fit(Xtrain,ytrain)
58. ypred = classifier.predict(Xtest)
59. accuracy = classifier.score(Xtest,ytest)
61. """
62. plots for comparing true labels to predicted labels
63. """
64. #plt.xlabel(Xtest.columns[0])
65. #plt.ylabel(Xtest.columns[1]+' time')
66. #scatterplot = plt.scatter(Xtest.iloc[:,0],Xtest.iloc[:,1], c = ypred)#prediction data
67. #scatterplot = plt.scatter(Xtest.iloc[:,0],Xtest.iloc[:,1], c = ytest) #true data
68. """
69. plt.xlabel(data.columns[0])
70. plt.ylabel(data.columns[1]+' time')
71. scatterplot = plt.scatter(data.iloc[:,0],data.iloc[:,1], c = labels)
72. plt.show()
73. """
75. """
76. Bivariate gaussian estimator
77. """
78. class0 = data[labels==0]
79. class1 = data[labels==1]
80. mean0,mean1 = np.mean(class0),np.mean(class1)
81. std0,std1 = np.std(class0),np.std(class1)
82. rho0 = np.corrcoef(class0,rowvar = False)[0,1]
83. rho1 = np.corrcoef(class1,rowvar = False)[0,1]
84. X0,X1,Y0,Y1 = np.linspace(mean0[0]-2,mean0[0]+2,50),np.linspace(mean1[0]-2,mean1[0]+2,50),np.linspace(mean0[1]-2,mean0[1]+2,50),np.linspace(mean1[1]-2,mean1[1]+2,50)
85. X0,Y0 = np.meshgrid(X0,Y0)
86. X1,Y1 = np.meshgrid(X1,Y1)
87. Z1 = (((X1 - mean1[0])\*\*2)/std1[0]) + (((Y1 - mean1[1])\*\*2)/std1[1]) - ((2\*rho1\*(X1-mean1[0])\*(Y1-mean1[1]))/(std1[0]\*std1[1]))
88. P1 = (1/(2\*np.pi\*std1[0]\*std1[1]\*(np.sqrt(1-rho1\*\*2))))\*np.exp(-Z1/(2\*(1-rho1\*\*2)))
90. Z0 = (((X0 - mean0[0])\*\*2)/std0[0]) + (((Y0 - mean0[1])\*\*2)/std0[1]) - ((2\*rho0\*(X0-mean0[0])\*(Y0-mean0[1]))/(std0[0]\*std0[1]))
91. P0 = (1/(2\*np.pi\*std0[0]\*std0[1]\*(np.sqrt(1-rho0\*\*2))))\*np.exp(-Z0/(2\*(1-rho0\*\*2)))
92. """
93. plot for class0
94. """
95. fig = plt.figure()
96. ax = fig.add\_subplot(111, projection='3d')
97. surf = ax.plot\_surface(X0,Y0,P0)
98. contour = ax.contourf(X0,Y0,P0,offset=-10)
99. scatter = ax.scatter(class0.iloc[:,0],class0.iloc[:,1],-10,c = 'blue')
100. ax.set\_zlim(-10, 5)
101. ax.set\_zlabel('probability density')
102. ax.set\_xlabel(Xtest.columns[0])
103. ax.set\_ylabel(Xtest.columns[1]+'time')
104. plt.show()
106. """
107. plot for class1
108. """
110. fig = plt.figure()
111. ax = fig.add\_subplot(111, projection='3d')
112. surf = ax.plot\_surface(X1,Y1,P1)
113. contour = ax.contourf(X1,Y1,P1,offset=-10)
114. scatter = ax.scatter(class1.iloc[:,0],class1.iloc[:,1],-10,c = 'yellow')
115. ax.set\_zlim(-10, 5)
116. ax.set\_zlabel('probability density')
117. ax.set\_xlabel(Xtest.columns[0])
118. ax.set\_ylabel(Xtest.columns[1]+'time')
119. plt.show()

Using the theory mentioned above we plot the following Gaussian distribution using our estimators, added are contour maps with the distribution of points for ease of observation.

(i) Blue labels - long eruption times cluster – class0:



(ii) Yellow labels - short eruption times cluster – class1:



**(e)MAP and Bayesian Estimates:**

**Solution 2: Applications of Inception module:**

**(a) Advantages:**

For a general CNN we need to decide at each layer what kind of filters we need to use, which primarily involves selecting the side of the filter. Some filters may perform better than other and it might be difficult to cross check each filters’ effectiveness as the upcoming and previous filters used may influence the current filter when back propagating. We therefore try to model an architecture which not only implements different sizes of filters but also involves stride-1 pooling layers along with convolution filters and we let back propagation decide which of them is to be used best and most.

**(b) Improvements:**

More filters obviously mean a lot more computations and weights to go through. The improvement of the inception module with dimensionality reduction tries to address this problem by reducing the depth of the inputs (or dimensionality) by applying 1x1 filters which are very fast to the inputs and then using larger convolution filters to get the output.

This dimensionality reduction is a very important factor, let us look at the number of operations for 128, 5x5x256 filters over an input layer of 28x28x256 (basic MNIST). With padding we are looking at,

5x5x256x28x28x128 = 642,252,800 Operations

Let us compute the number of operations for dimensionality reduced model,

Reducing the dimensionality by 4 with 1x1 filters, which reduces our input to 28x28x64,

1x1x256x28x28x64 = 12,845,056 Operations

We now apply the 5x5x64 filters to the reduced input

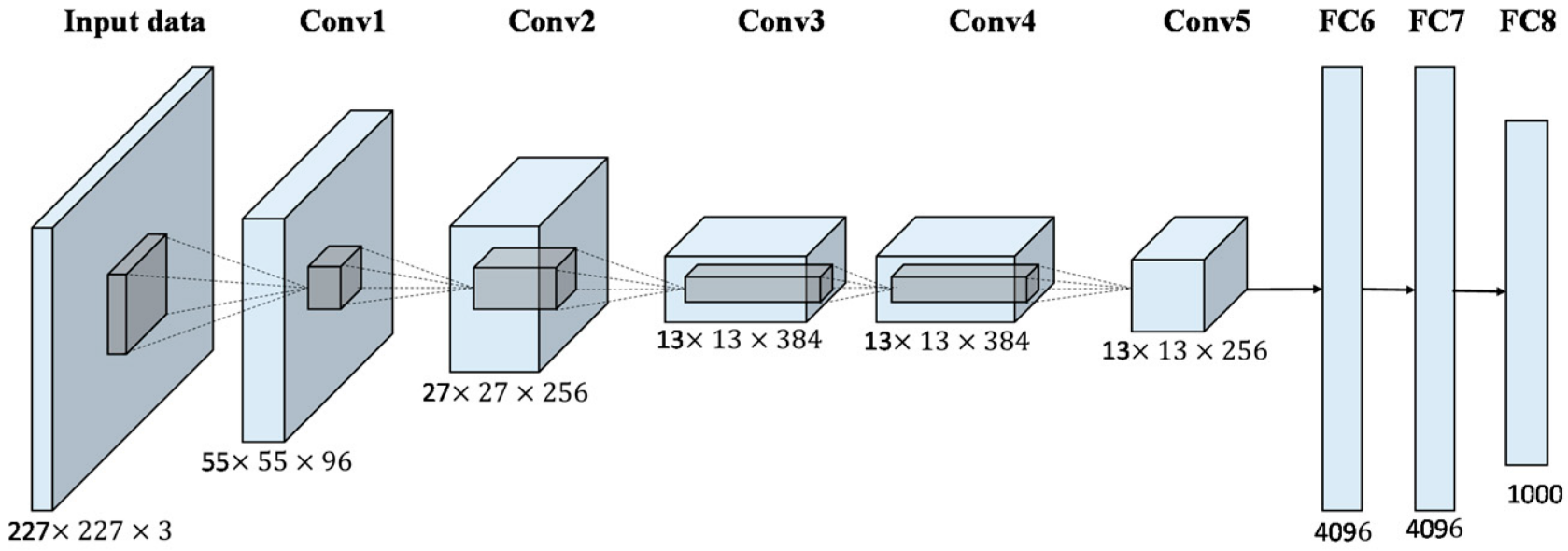
5x5x64x28x28x128 = 160,563,200 Operations

Total operations for reduced inception module = 173,408,256 Operations

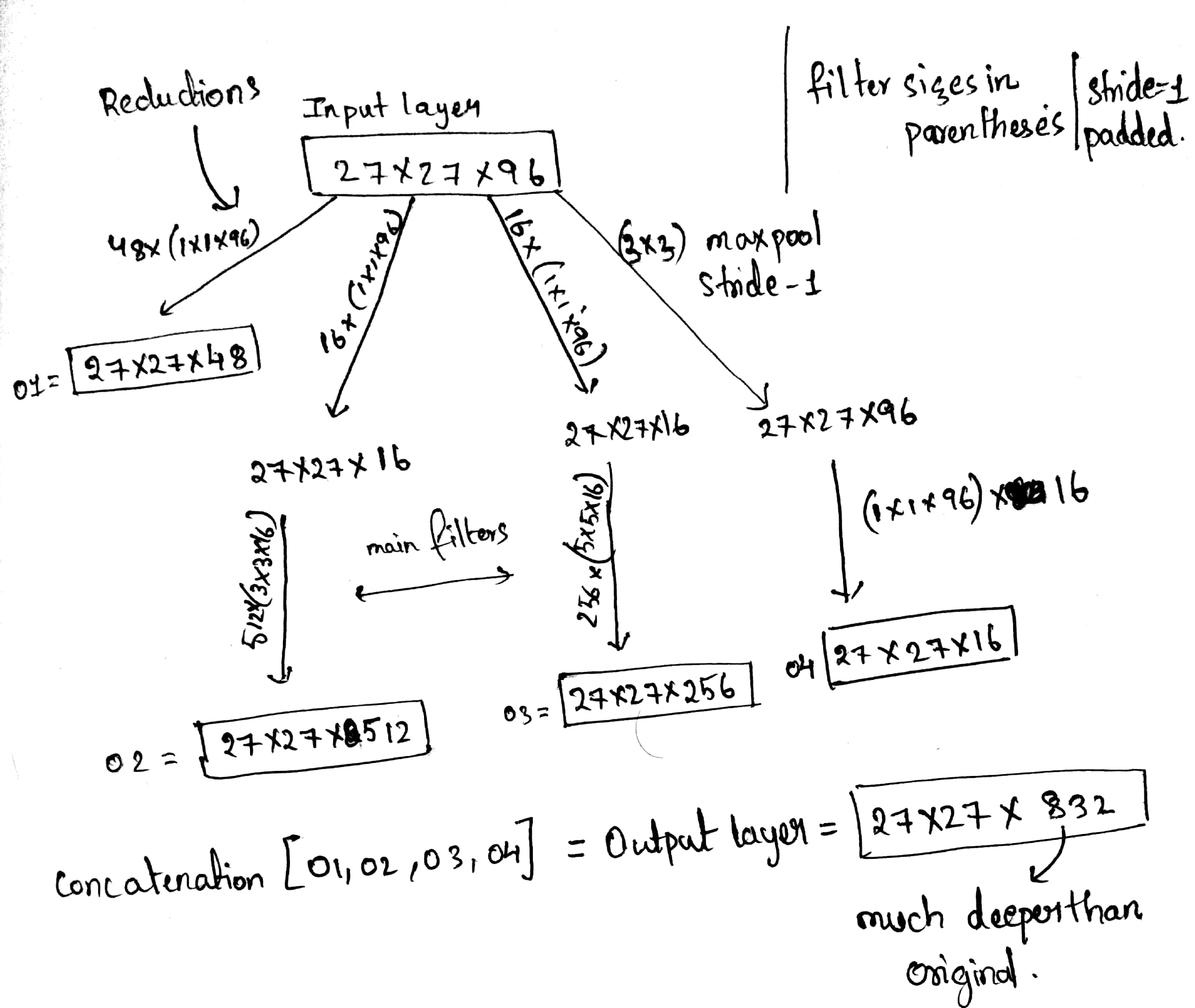
We see about ¼ reduction in the number of operations for the reduced model. This might seems a paltry improvement but that is only because of the very small reduction we are using. Higher orders of reduction give much better improvement in the computational performance with almost better than an order improvement over the naïve model when the reduction is increased to 1/16.

**(c)Modification over AlexNet:**

Let us look at the alexnet architecture in the image given below:



Let us try to create and dimensionality reduced inception module to the normalized, max pooled output of Convolution layer 1 (27x27x96) and apply convolution layer 2 and obtain a (27x27x256). This step uses 5x5 filters in the AlexNet, which we shall modify to get a much deeper image. We employ filters in the same ratios as applied in the GoogleNet architecture.



**(d)Application in Encoder-Decoder Network:**

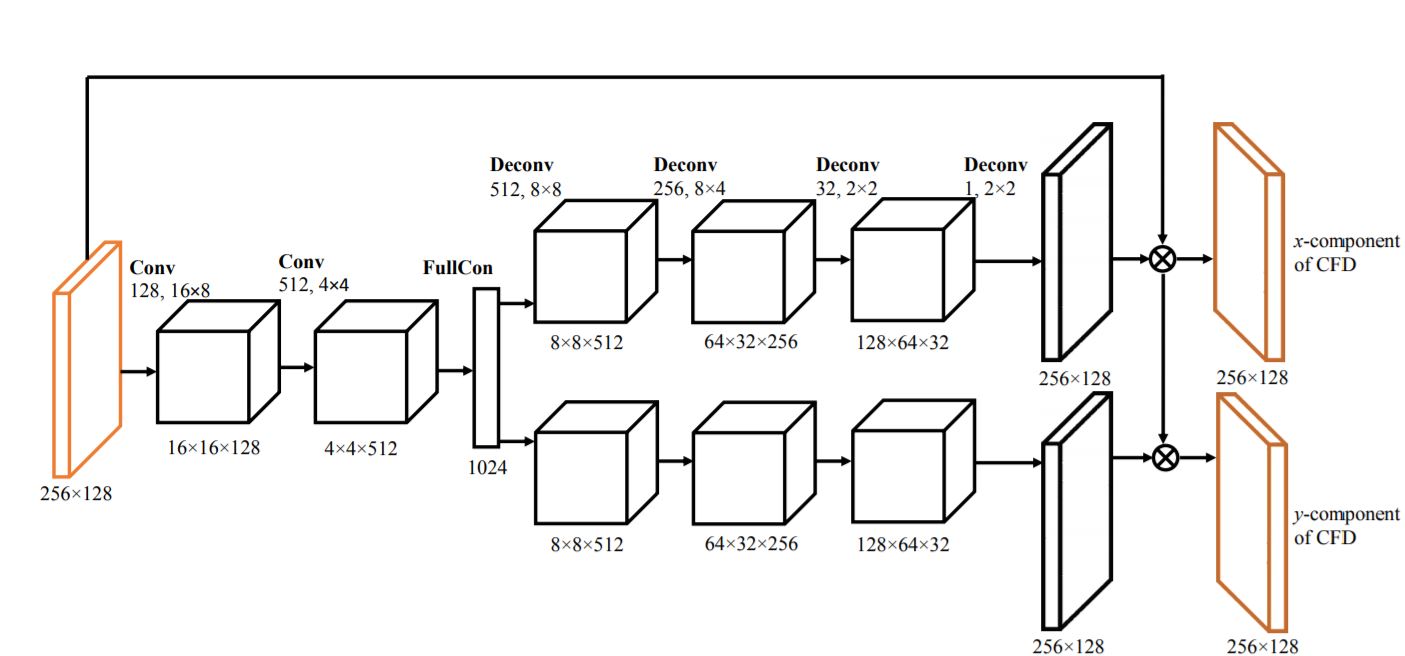
The primary use of encoder-decoder networks is to estimate continuous or seemingly unrelated variables, which is not a use case for a generic CNN.

Its major applications include Machine translations from one language to another as the languages have totally different construction rules and the final encoded vector helps in gauging the core meaning of a particular phrase or word which will then be converted to that another language with the decoder network.

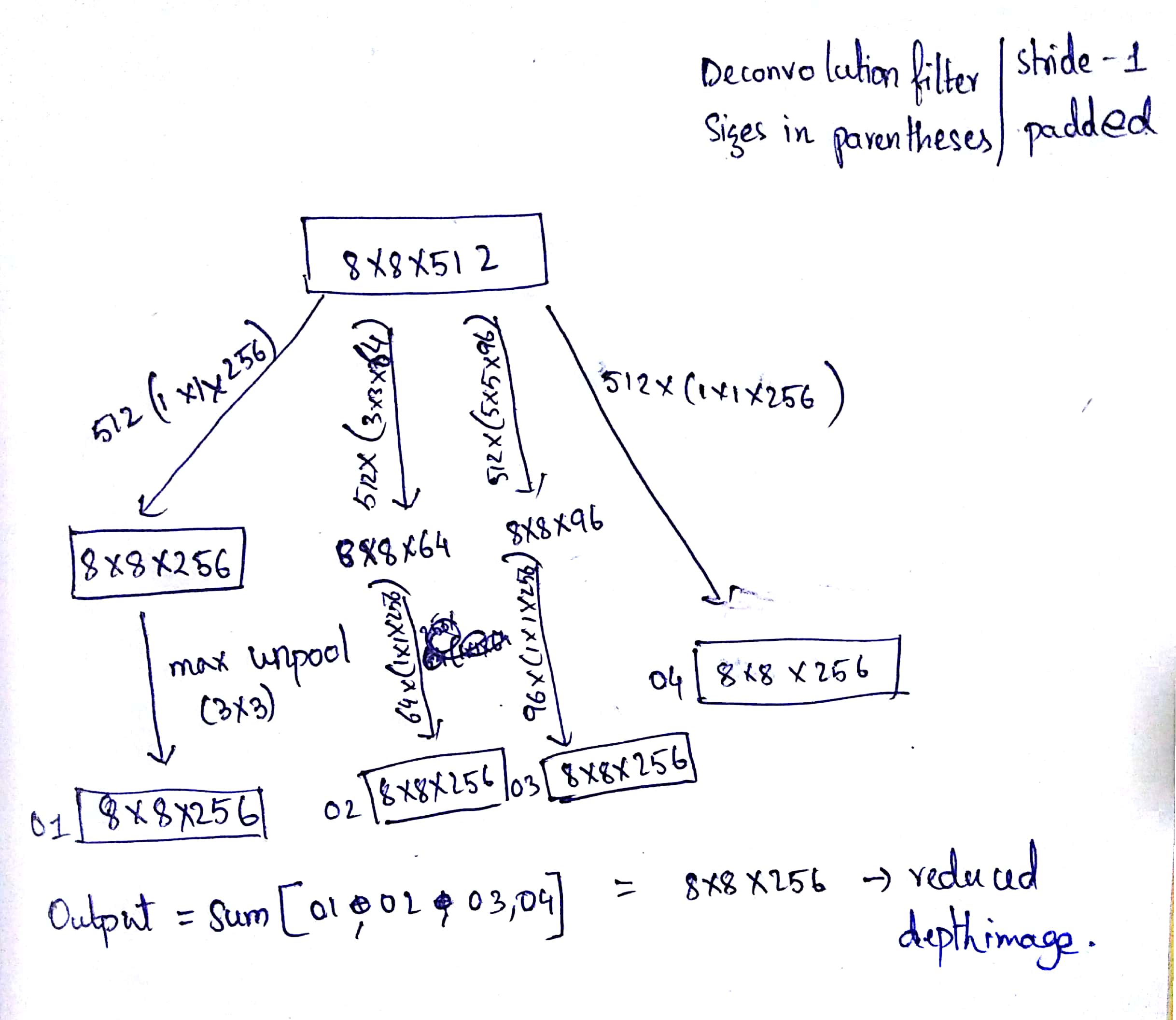
It also sees applications in CFD where it is proven to be orders faster than traditional methods like Lattice Boltzmann method when computing continuous vector fields of the flow.

**Ref:** 1. <https://arxiv.org/pdf/1409.3215.pdf>

2.<https://autodeskresearch.com/publications/convolutional-neural-networks-steady-flow-approximation>

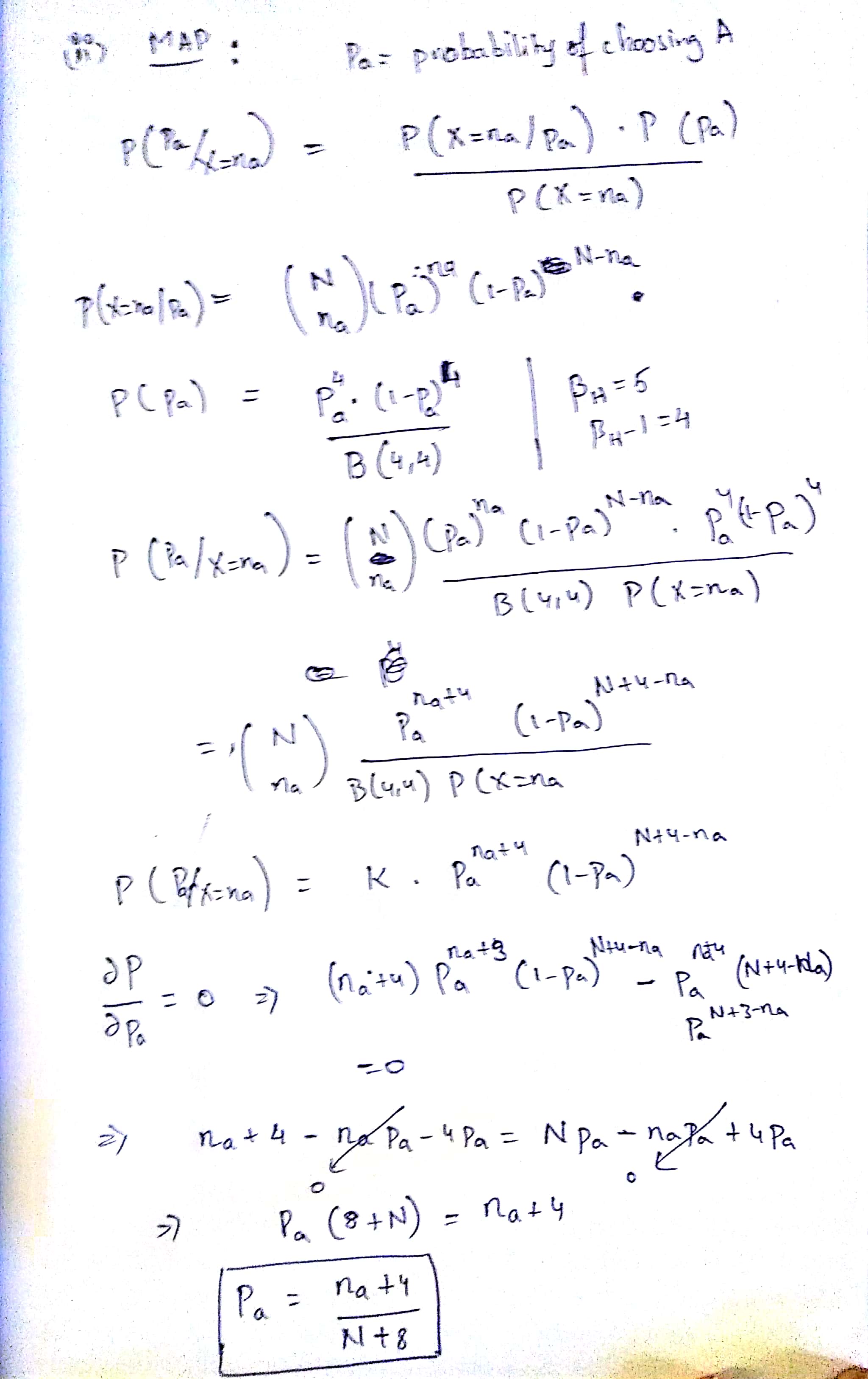


Implementation of the Inception module is conceptually the same as in the normal CNNs when dealing with the encoder side, the decoder side however is a total inverse of the CNN implements which is drawn below. The example used is the second Deconvolution layer from the image above,

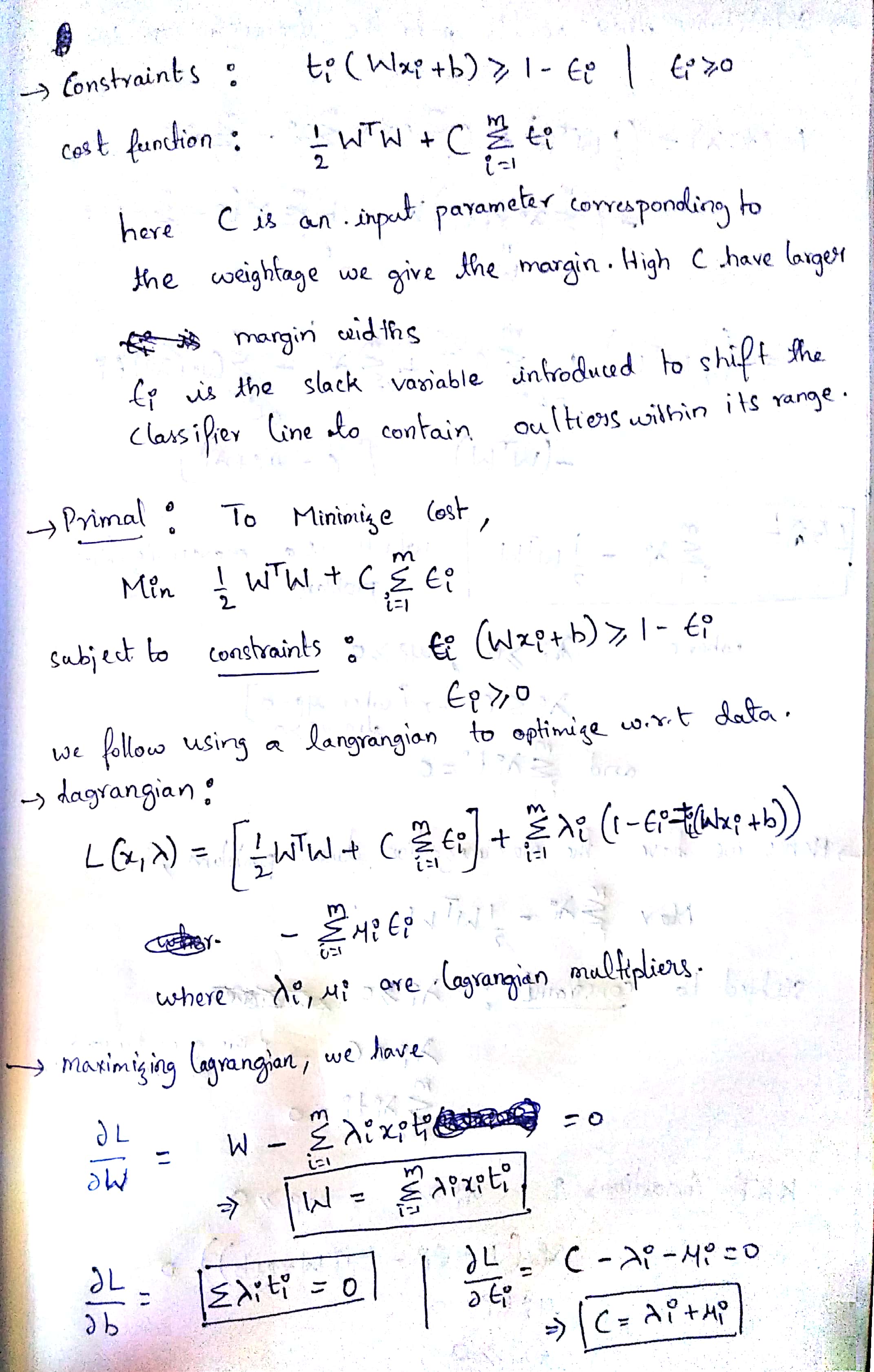


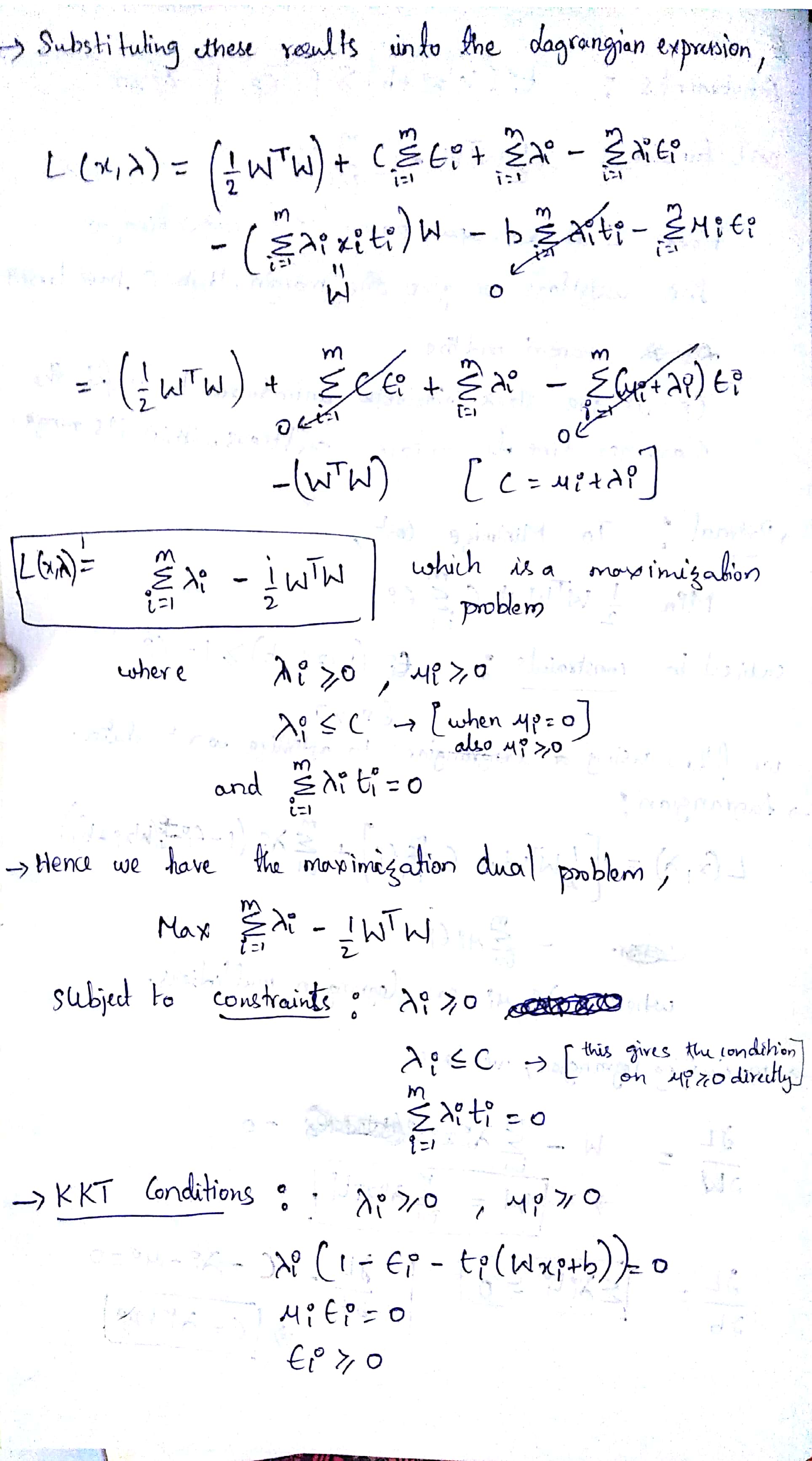
**Solution 4: Product preference:**

**Ref:** [**http://www.cs.cmu.edu/~aarti/Class/10701\_Spring14/slides/MLE\_MAP\_Part1.pdf**](http://www.cs.cmu.edu/~aarti/Class/10701_Spring14/slides/MLE_MAP_Part1.pdf)

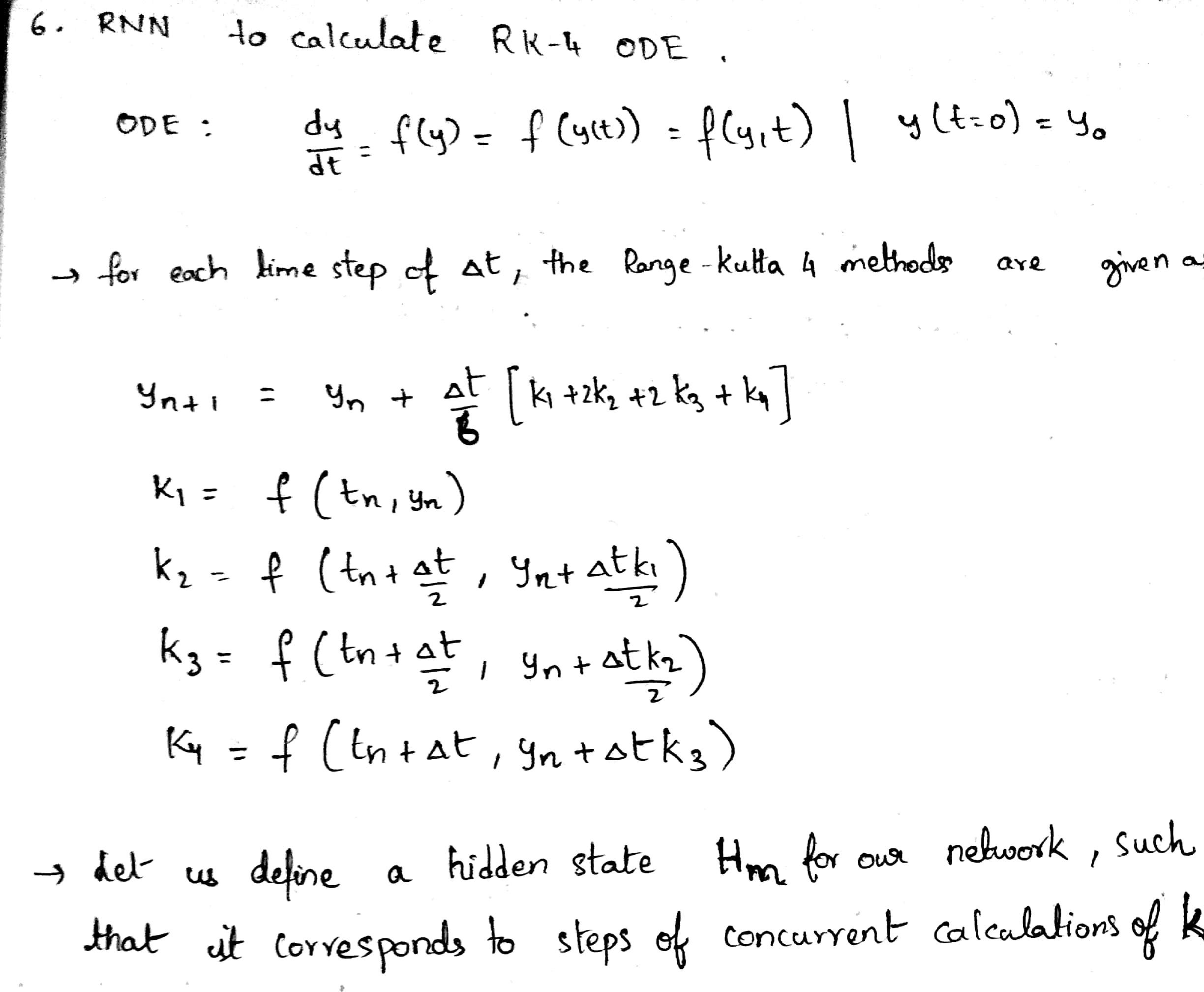
****

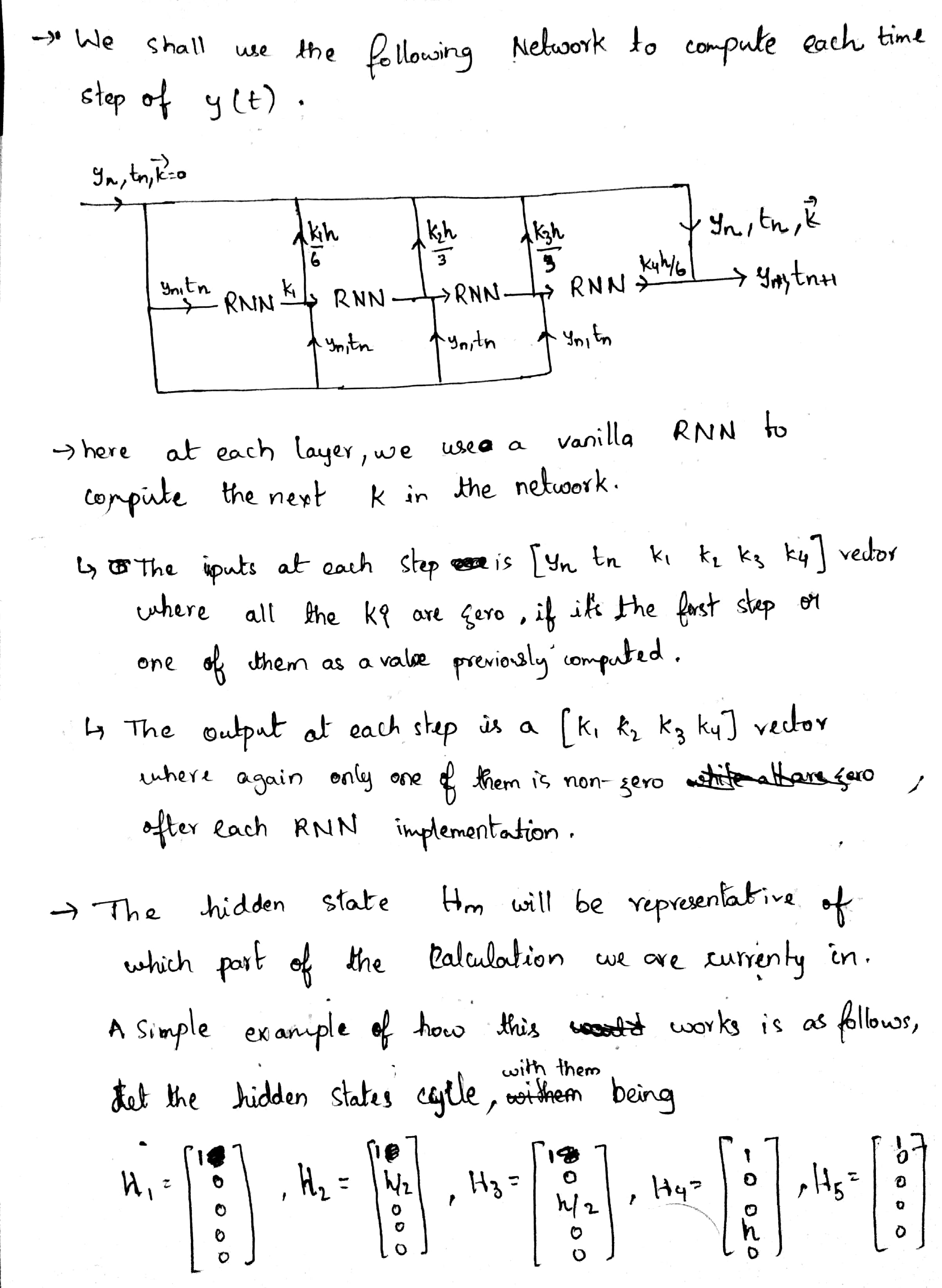
**Solution 5: Soft Margin Classification:**

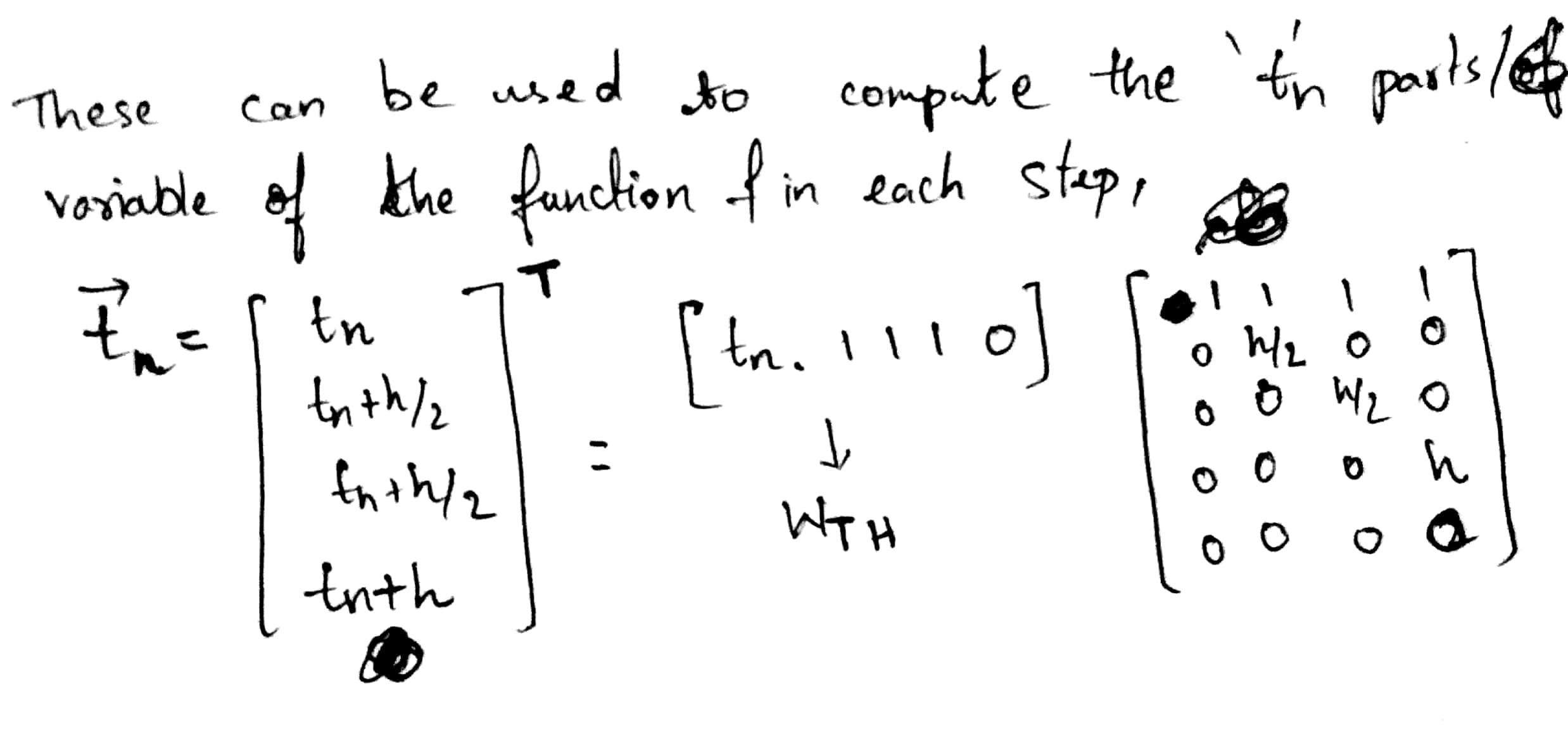
****

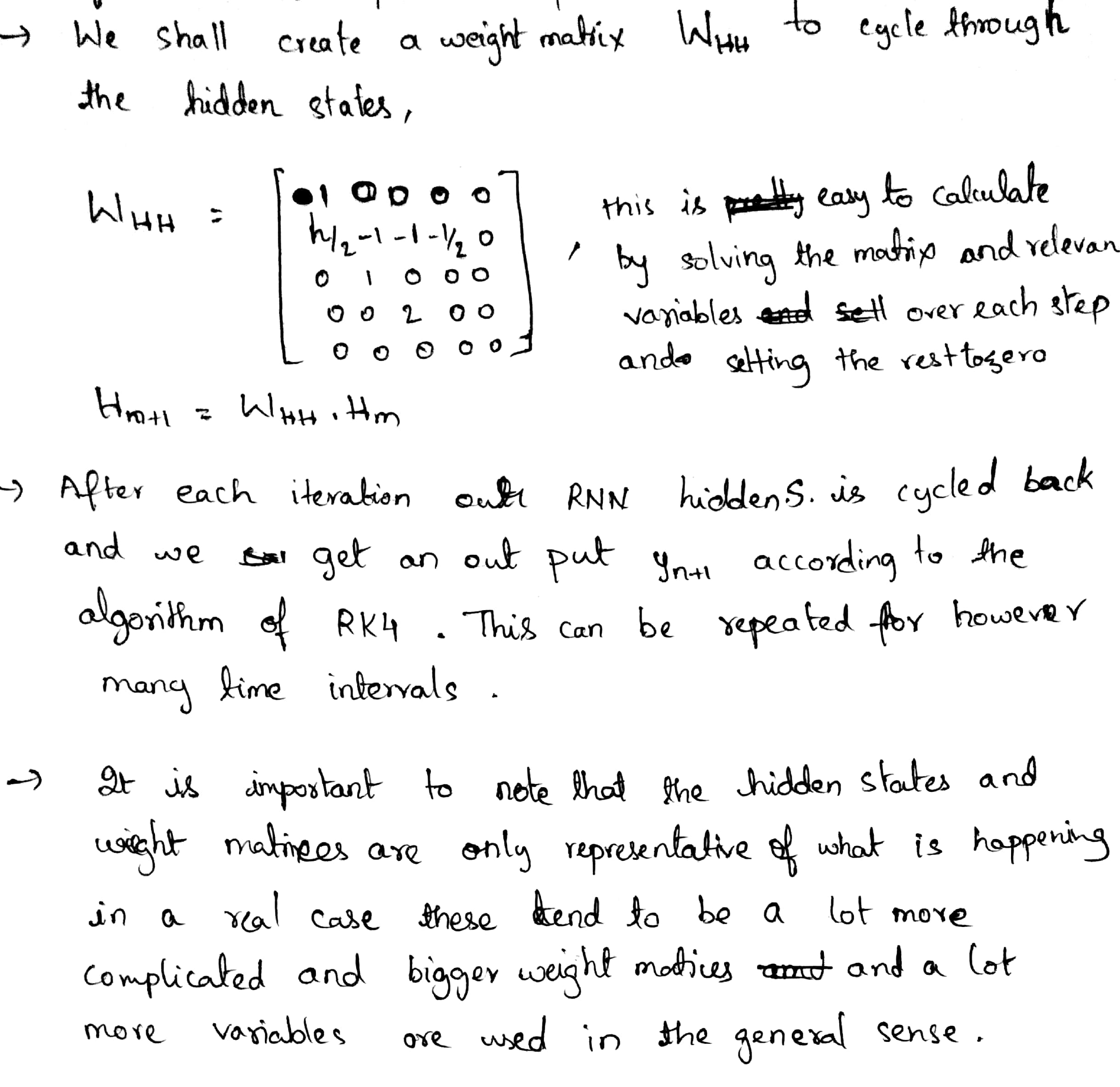
****

**Solution 6: RK-4 RNN implementation:**

****

****

****

****