Report on Cardiac Arrest Prediction

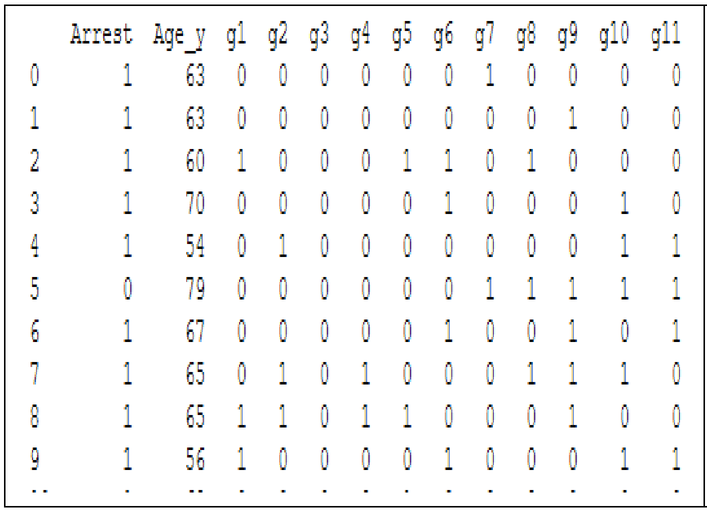
Git Link to the jupyter notebook code: https://github.com/kushalviit/Lucid_ckts_assignment

Tasks 1:

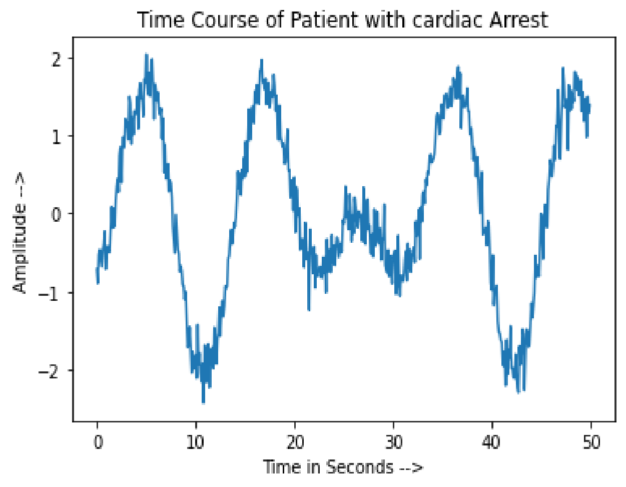
Visualize both data sets to highlight the information available in each data source, as well as potential associations of important variables/features with cardiac arrest.

The dataset was loaded as Pandas data frame.

- As part of initial understanding first few samples of Age and Mutation data was printed for understanding and a sample of the time course signal was plotted.

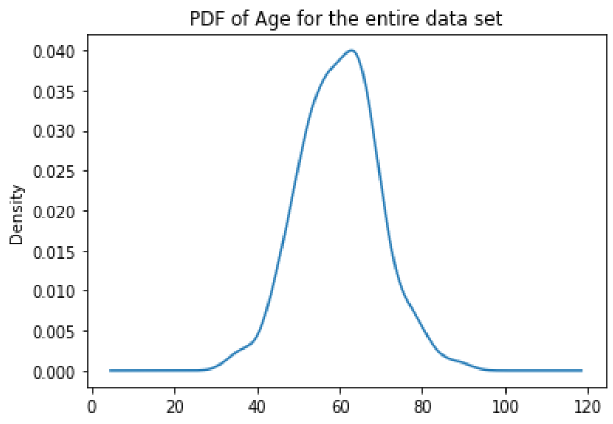


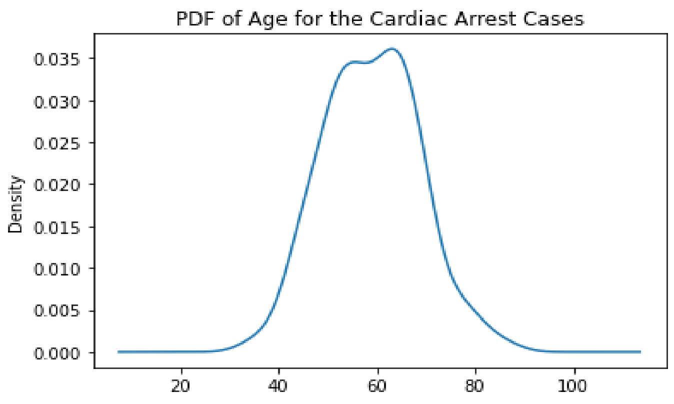
```
   Arrest  Age_y  g1  g2  g3  g4  g5  g6  g7  g8  g9  g10  g11
0       1     63   0   0   0   0   0   0   1   0   0    0    0
1       1     63   0   0   0   0   0   0   0   0   1    0    0
2       1     60   1   0   0   0   1   1   0   1   0    0    0
3       1     70   0   0   0   0   0   1   0   0   0    1    0
4       1     54   0   1   0   0   0   0   0   0   0    1    1
5       0     79   0   0   0   0   0   0   1   1   1    1    1
6       1     67   0   0   0   0   0   1   0   0   1    0    1
7       1     65   0   1   0   1   0   0   0   1   1    1    0
8       1     65   1   1   0   1   1   0   0   0   1    0    0
9       1     56   1   0   0   0   0   1   0   0   0    1    1
..    ...    ...  ..  ..  ..  ..  ..  ..  ..  ..  ..  ...  ...
```

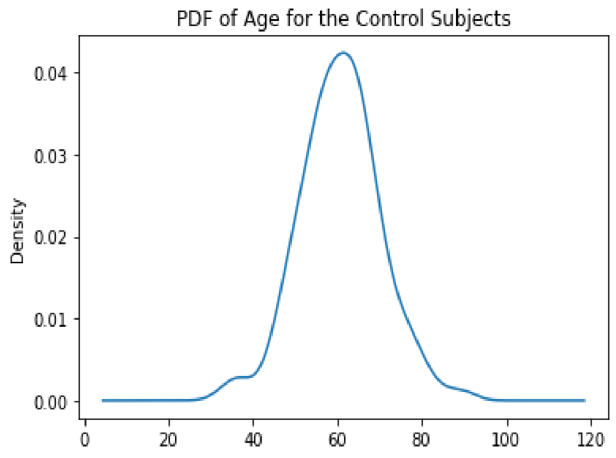


- **It was observed that there were 180 examples/samples/datapoints associated with Cardiac arrest. It was observed that there were 220 examples/samples/datapoints associated with Control Subject, with a total of 400 datapoints.**

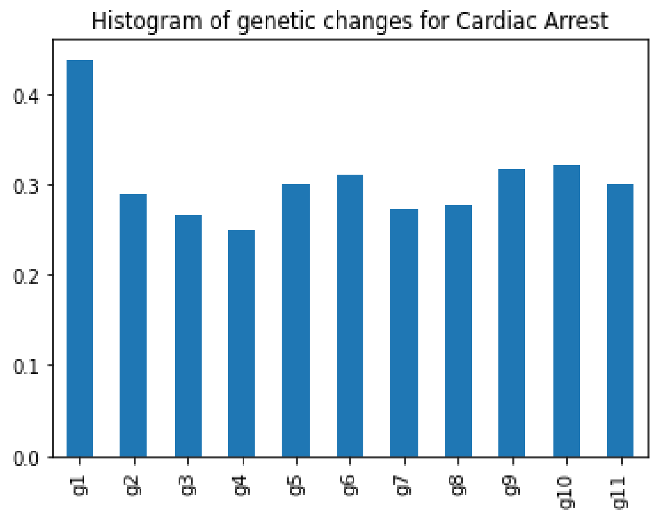
- The PDF of age in cardiac arrest subjects, control subjects and over all data points is shown below.
  Inference: The distribution of age over all data points has gaussian distribution indicating that <u>more data points have been</u> collected from individuals between the age of 40 and 80 while the rest of data points collected for other ages are less and this <u>correlation</u> can be seen in Cardiac arrest PDF and Control Subject PDF. *<u>Thus the feature Age_y has less/biased information to contribute for classification.</u>*

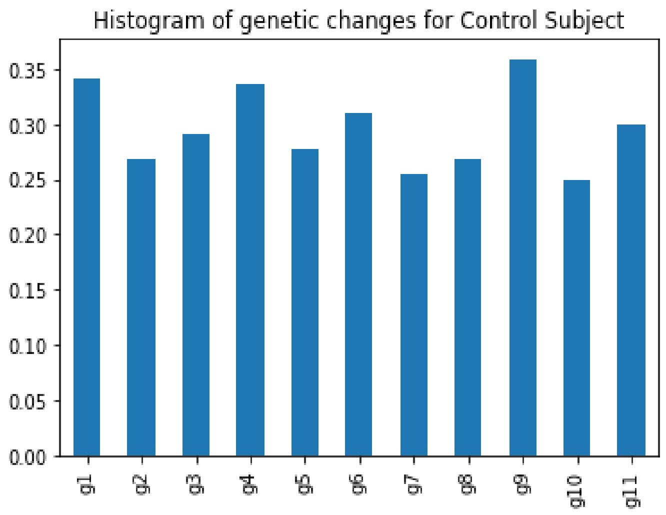


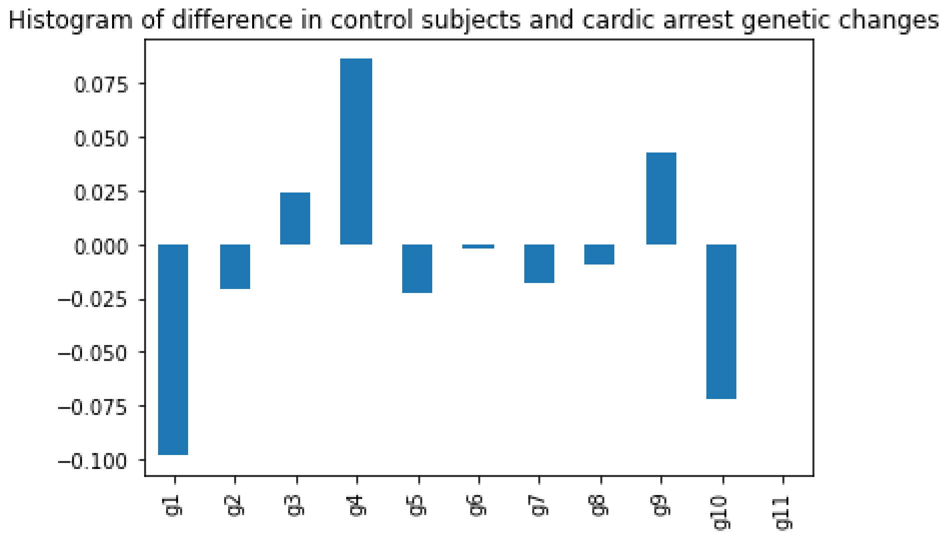




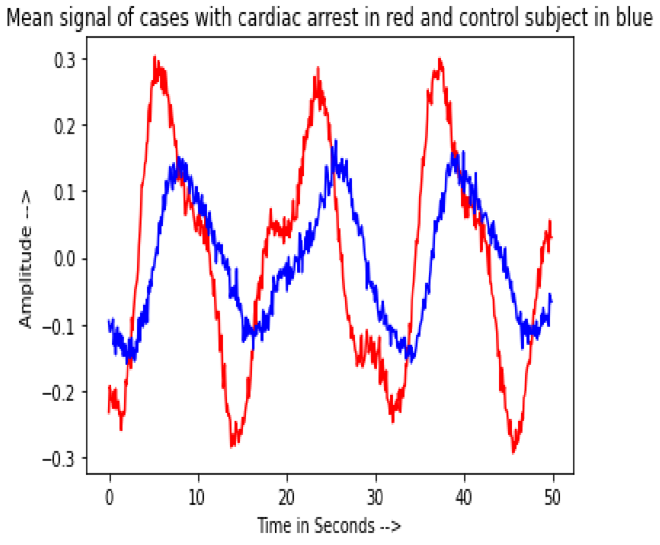
- Histogram of genetic mutations across Cardiac arrest subjects and Control subjects are as shown in the figure below **but it does not tell which genetic mutation or lack of mutation contributes to Cardiac arrest. Therefore, the difference in Histograms are plotted in the third Histogram. [Histogram has been normalized in both cases]**
  Inference: ***The Difference in Histograms(Control Subject Histogram – Cardiac Arrest Histogram) graph indicates that among all the data points g1 and g10 mutations in cardiac arrest cases are more when compared to Control subjects and small group of people with cardiac arrest have g2 , g5, g7 and g8 mutations when compared to control subjects. G3 g4 and g9 mutations are more common in control subjects. These differences are relative in nature as there are cases of mutations among all genes in both Control subjects and Cardiac Arrest Subjects.***

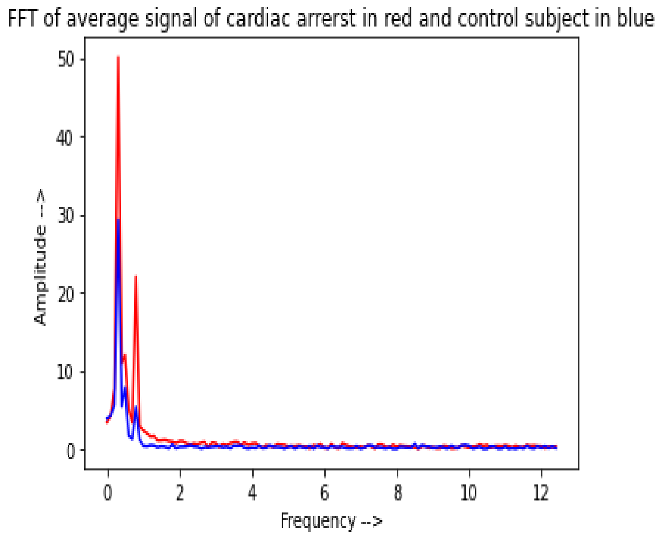






- Mean value across individual sample position of both Cardiac Arrest and Control Time-Course sample are plotted in the image below.
  Inference: If the time course samples were collected with the identical settings and sensors then there is a clear difference between the mean signal in terms of amplitude and variance. But some deviations and bumps other than normal harmonics are clearly visible and FFT shows that amplitude between 2 and 4 marking on FFT is higher than control cases





Tasks 2:

Develop a strategy to predict cardiac arrest from time-course signal recordings as well as other phenotypical information using at least two alternative methods. Make sure to highlight how you approach the problem in detail and how you go about validating your prediction strategy.

Data Set:

80% of the data is used for **training** with both classes containing equal number of examples (**160 cardiac arrest and 160 control subject)** and the remaining is used for testing and yes the **test dataset is biased** as there are more examples of control subject than test subject.
On training the training data is again sub divided into batches of size 64 and the rest of the training data can be used for validation by setting a validation as true in the functions of Keras.

Model 1:

- **Overview:**
  Part of the features are sparse vectors of containing 1s for mutation and other features include Age, Time course mean and Std Deviation of each signal. The sparse vector can be used to extract embedding vector which can provide relationship between genetic mutation. The rest of the features can be concatenated with embedding vectors in order to be used with a classifier. In this case, the classifier chosen is Fully connected Neural Network.
- **Features:**
  1. *Transformed Genetic Mutation*, *Normalized Age* [Age/100], Time Course *Mean* and *Std deviation* across entire signal of 500 samples are the four features.
  2. *Transformed Genetic Mutation* involves transforming sparse 0,1 codes to Dense code. Element wise Vector multiplication is conduced on the data point and the values greater than zero are the dense codes of Genetic Mutation.
     Example: Let [0,1,0,1,0,0,0,0,0,1,0] is the *Genetic Mutation vector* representing [g1,g2,g3,g4,g5,g6,g7,g8,g9,g10,g11] the respective genetic mutation.

The *reference dense code vector* is given by [1,2,3,4,5,6,7,8,9,10,11].
On **element wise vector multiplication and elimination of 0 s** in the resultant vector the dense vector is obtained which for the given example is [2,3,10]

3. The above transformation creates a dense vector of varying length and the architecture of the ML used needs a fixed vector hence **padding with zeros** is used as solution.
(*This method reduces the vector length of sparse vector from 12 to dense vector 7*)

- **Architecture:**
The model used consists of embedding layer which uses a vocabulary size of 64 with each embedding vector of length 4. Since the input to embedding layer is a vector of length 7 the output is a 2-D vector of shape 7x4 which is flattened as a vector and concatenated with three other features [Totaling the Feature Vector length to 31]. These features are given as input to Dense/Full Connected Layer with output of size 10 with 'RELU' activation and finally for the output there is another Dense layer of output size 1 with 'Sigmoid' as activation function. The entire architecture is summarized in the Keras Generated table below.

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 7)             0
____________________________________________________________________________________________________
embedding_1 (Embedding)          (None, 7, 4)          256         input_1[0][0]
____________________________________________________________________________________________________
reshape_1 (Reshape)              (None, 28)            0           embedding_1[0][0]
____________________________________________________________________________________________________
input_2 (InputLayer)             (None, 3)             0
____________________________________________________________________________________________________
concatenate_1 (Concatenate)      (None, 31)            0           reshape_1[0][0]
                                                                   input_2[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 10)            320         concatenate_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 1)             11          dense_1[0][0]
====================================================================================================
```

- **Results/Discussion:**
The training accuracy and testing accuracy both were observed to be around 60% indicating that there is on overfitting. The model is stuck in a local optimum. To move ahead further feature analysis can be done.
Confusion matrix can be used to further analyze the predictions and is shown here

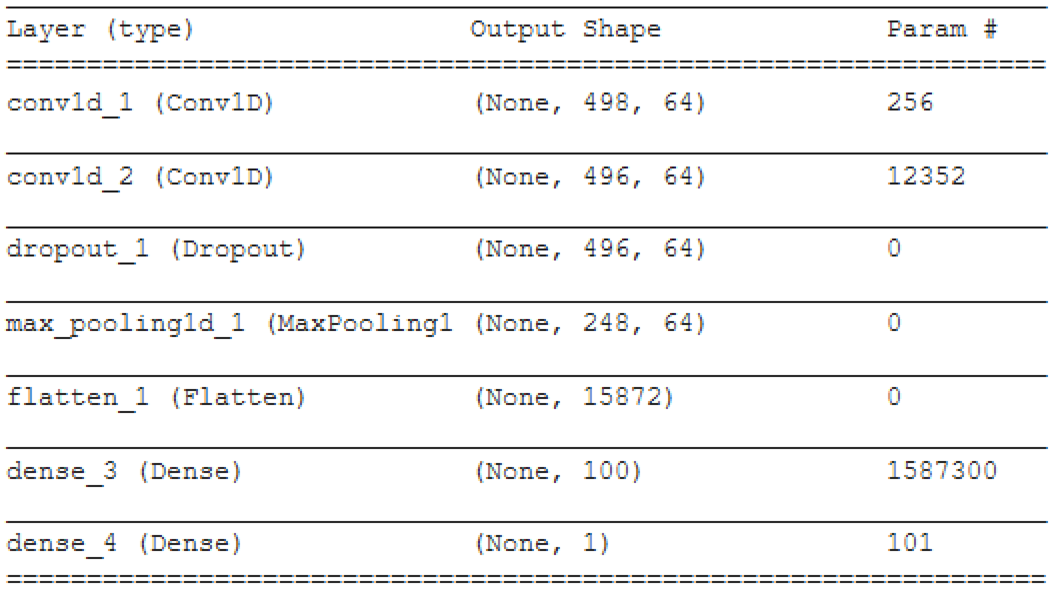
	Predicted as Class 0(Control subject)	Predicted as Class 1(Cardiac arrest)
Actual Class 0(Control subject)	40	20
Actual Class 1(Control subject)	9	11

So 66% of the control subjects are identified as control subjects while 55% of the cardiac arrests are identified as cardiac arrest miss cases of identification of cardiac arrest are 45% which is really dangerous and life threatening while 34% of the control subjects are identified as cardiac arrest patients which has bad effects but the effects are not life threatening

Model 2:

- **Overview:**
The Time course signal of shape 500x1 can be subjected to 1-D convolution layers and the output can be flattened and combined as features and can be used with a fully connected network for classification.

- **Features:**
  The unaltered Time-Course Signal of length 500.
- **Architecture:**
  The architecture used here consists of 1 D convolution with input shape of 500x1 with an output of 64 channels and kernel size of 3 without padding on input and default stride of 1. This output is again subjected to 1D Convolution with same parameters as before and this is followed by 1D Maxpooling followed by Fully connected layer as shown below by Keras layers output.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 498, 64)           256
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 496, 64)           12352
_________________________________________________________________
dropout_1 (Dropout)          (None, 496, 64)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 248, 64)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 15872)             0
_________________________________________________________________
dense_3 (Dense)              (None, 100)               1587300
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 101
=================================================================
```

- **Results/Discussion:**
  The training accuracy is about 86% and testing accuracy is about 75%. This model seems to be better but there are hints of overfitting as the difference between training accuracy and testing accuracy is huge.

	Predicted as Class 0(Control subject)	Predicted as Class 1(Cardiac arrest)
Actual Class 0(Control subject)	47	13
Actual Class 1(Control subject)	7	13

The control subject identification is at 78% and the cardiac arrest identification is at 64% which is slightly better than Model 1 but needs to be improved.
One way to improve is to use more convolution layers so that the identification capability increases or use **multifrequency signals using DWT** to improve the results.

Tasks 3 and 4:

Provide an analysis of the computational cost associated with the learning strategies adopted in the previous point.
Provide an analysis of the computational cost associated with deployment of the models used in 2. for prediction of cardiac arrest in new patients

Solution for Task 4: It is fair to **Consider only Forward Propagation costs** for **deployment of both models** in **new patients for prediction and batch_size =1 if individual patients are considered for real time deployment**.

Analysis:

Learning involves both Forward and Backward Propagation in both the models and the analysis of Computational costs are as shown below:

Model 1:

For each iteration in an epoch there is a forward propagation, Computation of loss function and backward propagation [Some of the analysis done here are based on assumption for example implementation of sigmoid function but sigmoid function can also be implemented by other ways]

For each Forward Propagation

*ONLY Multiplications and addition costs are being considered as major costs and other operations are neglected for this analysis

* For Forward operation W and b are fixed and need not be loaded every time if there is enough available memory in immediate Cache or RAM for all W and b

*batch_size=64

* Assuming exponential implementation using ***Taylor series with 3rd order approximation***

[e^-x =1-x+(x^2/2)-(x^3/6)]

Layers in Model 1	Mathematical Function	Computation	Memory	Datatype
Embedding Layer	Loop up table	0	Look up table 64x4; Input 7x1xbatch_size	Float
Dense 1	Y2=W2*X2+ b2	Multiplications:10x31xbatch_size; Additions: 10x30xbatch_size +10xbatch_size	W2:10x31; b2:10x1; X2: 31xbatch_size	Float
Activation	Y3Relu (Y2>0=Y1 else 0)	0	10x batch_size	Float
Dense 2	Y4=W4*Y3+b4	Multiplications:1x10xbatch_size; Additions: 9xbatch_size +1xbatch_size	W_4:1X10; b_4=1x1 X_4=10xbatch_size	Float
Activation	Sigmoid(1/(1+e^-x))	For Tylor Series alone [Addition/subtraction: 3xbatch_size Multiplication/Division:7xbatch_size] For entire Sigmoid [Addition/subtraction: 4xbatch_size Multiplication/Division:8xbatch_size]	Input:1xbatch_size Output:1xbatch_size	Float

For Each Loss Function:

The loss function used is binary cross entropy which is given by

m=batch_size

y’=output of sigmoid layer

L=-(1/m)* Sum{y*log(y’)+(1-y)*(log(1-y’)))}

Log(x) Taylor approximation (3rd order): mult/div=7 ; add/sub=2

Computation cost per example in batch_size: mult/div=7*2+2 ; add/sub=2*2+2

Total Computation cost of loss function per batch: mult/div=(7*2+2)*m +1; add/sub=(2*2+2)*m

For each Backward Propagation: [Assuming ADAM optimizer]
For each Dense Layer following computations are involved in adam optimizer [1]:

```
Loop until convergence:
    Step1: g_t=Gradient_of_each_function for each iteration
    Step2: m_t=beta_1*m_(t-1)+(1-beta_1)*g_t [First_Raw_moment]
    Step3: v_t=beta_2*v_(t-1)+(1-beta_2)*(g_t ^2) [Second moment]
    Step4: bias correction m_t=m_t/(1-beta_1^t)
    Step5: bias correction v_t=v_t/(1-beta_2^t)
    Step6: theta_t=theta_(t-1)- alpha* m_t/(sqrt(v_t)+epsilon)
```

m=batch_size
Step 1: involves gradient computations like

- Computation of d_L/d_y=-(1/m) sum(y/(y’(1-y’)))
  Computational Cost: Multiply /Divide=2*m+1 ; Add/sub:2*m

- Computation of gradient for sigmoid function with **chain rule:**

  d_s/d_y= sigmoid(Y4) * (1-sigmoid(y4))*d_L/d_y
  Computation cost: Multiply/Divide=10*m [reasoning 8 for sigmoid +1 for chain rule+1 for differentiation whole thing multiplied by batch size] ;Add/sub=5*m

- Computation of gradient for of Dense 2 with **chain rule**:

  d_Y4/d_W4=Y3*(d_s/d_y) [d_s/d_y is 1 x m vector and Y3 is m x 10 matrix multiplication of d_s/d_y]
  Computation cost: Multiplications=10*m
  Additions=9*m

  d_Y4/d_b4=Sum(d_s/dy) sum of all element of d_s/d_y [basically d_s/d_y dot product with vectors of 1]
  Additions=m

- Computation of gradient for of Activation function with **chain rule**:
  da/dy=Y_2’*W4*ds/dy
  where Y_1’ is a vector of 1’s and 0’s which computed using the Y_2 i.e, if Y_2 is positive then 1 else 0.

  Computational Cost:

A **element wise multiplication** of Y_2' with W4 is done and the resultant vector is multiplied **as element wise product** with ds/dy [W4 shape= 10x1 Y_2' shape= 10 xm and ds/dy shape=mx1]
According to the preceding logic
Multiplications=10*m*m

- Computation of gradient for of Dense 1 with **chain rule**:

  d_Y2/d_W2=X2*(da/dy) [da/dy is 10xm and X2 is mx 31 and matrix multiplication yields a matrix 10x31]
  Computation cost: Multiplications=10*m*31 Additions=(m-1)*10*31

  d_Y2/d_b2=Summation of da/dy along columns
  Computation cost: Additions=10*(m-1)
- Computation of gradient for of **flattened embedded encoding** with **chain rule**:
  d_Y1/d_y= W2.T*da/dy
  W2.T is the transpose of W2 and 31x10 multiplied with da/dy of size 10xm yielding a matrix of 31x m
  Computation cost: Multiplication: 10*31*m ; Additions=31*9*m
  Only first 28 features with m examples are related to embedding.

  Generic Computation table with respect to rest of the backward propagation is shown below:
  Assuming each parameter vector is of length n i.e first momentum vector of length n , second momentum of length n and gradient of length n. **The length of n can be visualized for each layer by flattening the parameters and gradient update is needed only for Dense layers (not needed for activation layers). [Flattening the parameters: example in Dense 1 Layer W2 is a matrix and can be flattened into a vector for update]**

Steps	Function	Computations
2	m_t=beta_1*m_(t-1)+(1-beta_1)*g_t	Mul/div=2*n Add/sub: n+1
3	v_t=beta_2*v_(t-1)+(1-beta_2)*(g_t ^2)	Mul/div=3*n Add/sub: n+1
4	m_t=m_t/(1-beta_1^t)	Mul/div= n+t+1 Add/sub=1
5	v_t=v_t/(1-beta_2^t)	Mul/div= n+t+1 Add/sub=1
6	theta_t=theta_(t-1)- alpha* m_t/(sqrt(v_t)+epsilon)	Mul/div=2*n Add/sub=n+1 *sqrt computation of individual components need extra time which is not indicated.

Model 2:

Only Forward Propagation is explained here, detailed computation cost analysis can be done as was done in case of Model 1. Activation layers are left out from analysis and analysis can be done in similar fashion as earlier model

Layers in Model 1	Mathematical Function	Computation	Memory	Datatype
Conv_1d	Y=W*X+b/Stride=1/Kernel=3/filters =64	Multiplication: 3*498*64*batch_size Addition: 3*498*64*batch_size (Addition per conv=2 +1 bias)	Bias b=64 W=3x64 Input X=500	Float

			Output:498x 64	
Conv_1d	Y=W*X+b/Stride=1/Kernel=3/filters =64	Multiplication: 3*496*64*64*batch_size Addition: 3*496*64*64*batch_size (Addition per conv=2 +1 bias)	Bias b=64 W=3x64x64 Input X=498x64 Output:496x 64	Float
Dense 1	Y=W*X+b	Multiplication:100*15728*batch_size Addition:100*15728*batch_size (includes bias)	W=100 x 15872 b=100 Input=15872 Output=100	Float
Dense 2	Y=W*X+b	Multiplication:100*1*batch_size Addition:100*1*batch_size (includes bias)	Input 100 Output 1	Float

Tasks 5:

For one of the learning strategies discussed in 2., discuss how one may modify the model to improve on computational efficiency in terms of both computational speed and memory usage.

For this Task I am using **Model1** for demonstration.

- From feature analysis done in Task 1. It can be seen that gene 5 6 7 and 8 are not contributing much as information for classification and from Task 2 Model 1 it was observed that with just the gene mutation information the accuracy was low indicating that the embedding is not showing connections between mutations Thus 5,6,7,8 genes can be eliminated.
- This in turn decreases the length of input from 7. As an example, if it reduces from 7 to 5 and if we reduce the embedding vector to 2 then the size of embedding lookup table gets reduced to 64x2 and the output of embedding layer gets reduced 5x2 vector per example. The reduction in vector size for Dense 1 Layer reduces from 7x4+3 to 5*2+3. This shows that the changes in algorithmic level can bring in significant reduction in memory. This also reduces computation as the matrix size is reduced and in turn the multiplications and additions of matrix are reduced.
- Using pipeline-based **vector-vector multiplier with accumulator [Shown in figure below]** reduces the number of multipliers and adders needed for the computation thus enabling utilization of less resources with increase in latency but as **trade off** requirement **for pipeline-based** memory increases.
- Sigmoid activation functions can be implemented as a look up table this **decreases computation** cost but **increases memory for lookup tables**
- Another way to speed up would be to switch from floating to fixed point data type.