Report

Name: Ying Xu
E-mail: xuying1991@hotmail.com
Last four digits of student ID: 9859

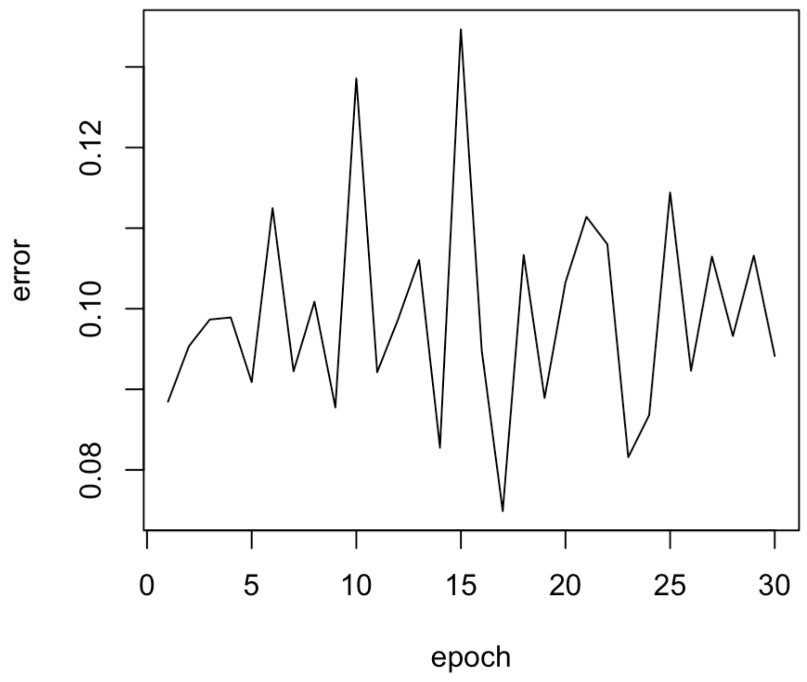
Question 1a:

The R built in function I used is neuralnet(). From last assignment, I chose first 16 predictors as input for neural network: RBC, PCV, Hb, HbS, MCH, MCV, Age, Nsex, BEN, Polys, WBC, Retic, BAN, SEN, CAM, HbF.

Delta rule is done by using backpropagation algorithm with single layer, sigmoidal activation function is defined by ‘logistic’, I also choose cross-entropy as the loss function since it’s better for binary classification, learning rate is set as 0.01 to avoid missing drop in error. The number of epoch is 30 to avoid overfitting. The training error value I got for each run with 10 runs are as below:

```
> error_vector
 [1] 0.10027626725 0.10678202439 0.10420787927 0.10364951140 0.10461997919 0.10077917437
 [7] 0.09991591624 0.09917533439 0.10478480651 0.10070837822
```

The best run is the 8th run with lowest error, and its error plot is shown below:

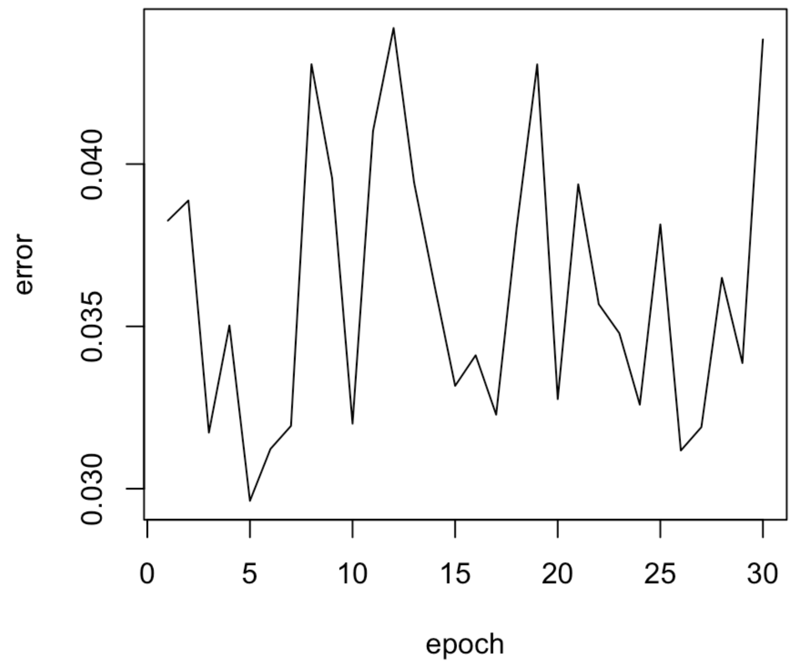


Question 1b:

Two-stage backpropagation is defined by using two layers c(8,4) backpropagation, other input parameters are the same as delta rule network. The training error value I got for each run with 10 runs are as below:

```
> error_vector_bp
 [1] 0.03806204243 0.03919378978 0.03611715780 0.04018242545 0.03783226084 0.03801709286
 [7] 0.03749392018 0.03862884991 0.03841419765 0.03768838451
```

The best run is the 3rd run with lowest error, and its error plot is shown below:



Question 1c:

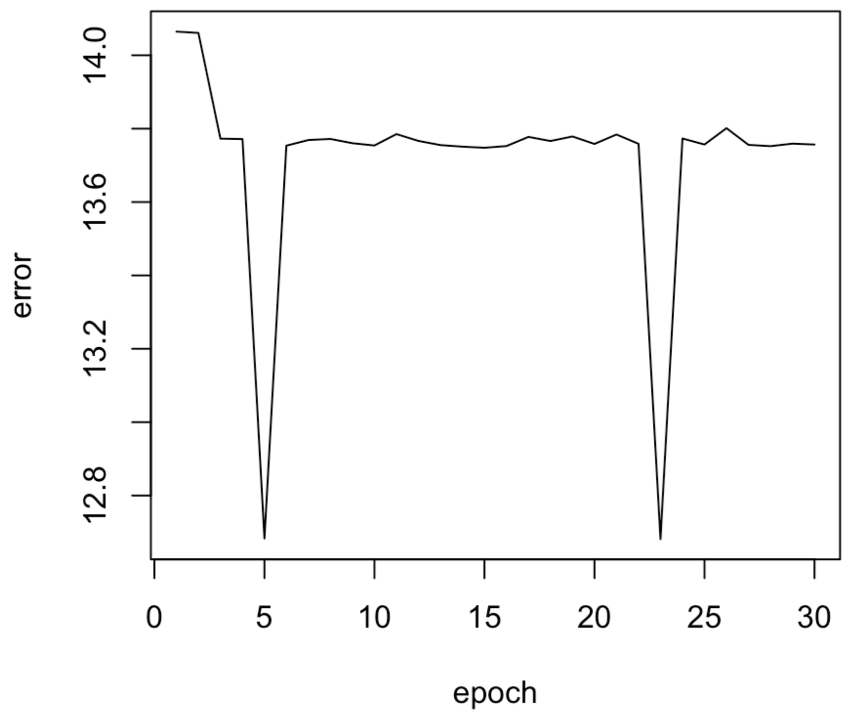
Comparing delta rule and two-stage backpropagation neural networks, the mean training error (cross-entropy) for delta rule is 0.1, and 0.04 for two-stage backpropagation. This is what I expected since two-stage backpropagation has more layers, the model will be able to learn more by adjusting weight and minimizing error.

Question2:

Delta rule neural network:

The best run is the 4th run with lowest error, and its error plot is shown below:

```
> error_vector_dr
 [1] 13.75344363 13.80698045 13.75809192 13.71271502 13.74294519 13.78882469 13.70071836
 [8] 13.78401582 13.79476246 13.74967304
```



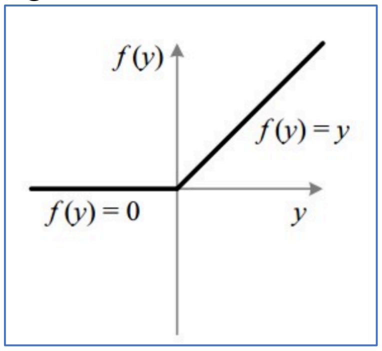
Two-stage backpropagation neural network:

This step takes very long to rue and it's still running when the assignment is due, I tried improving learning rate to 0.1 and stepmax to 1e+07, but there are still cases that the neural network cannot converge within the stepmax. But my assumption is that the error for two-stage backpropagation neural network will be smaller than single layer neural network.

Question 3b:

i. There are mainly two types of models available in Keras: sequential models and models created with the functional API. I used sequential models for Golub dataset because it' s good for binary classification and works for small dataset. Functional API is for defining complex models, such as multi-output models, shared layers models. Convolutional model is also a very popular way to do computer vision, image feature recognition.

ii. The activation function I used for middle layers is ReLU. ReLU is a type of non-linear activation function, it’ s a good fit for binary classification and the most commonly used one. The figure below shows that negative input will be convert to 0 and the neuron isn’ t activated, therefore makes computation easier and speeds up the training. Other kinds of activation function are not good fit to Golub dataset, such as linear activation function is not good at adjusting gradient so that errors could remain the same, sigmoidal and tanh fit better as an output activation function.



iii. First, I tried tanh activation function, but the error plot shows two parallel lines of training and cross validation. It could be data range and output range are not the same cause the model to be confused at prediction. Since I have normalized Golub dataset to be in range 0 to 1, the classification range is 1-2, however there is no activation function can adjust output data into this range, hence I subtract one from each value and get classification range in 0-1, then I tried sigmoid activation function which also has range 0 to 1.

<u>Question 3d:</u>

i. The loss function I used is binary cross-entropy. According to the formula of cross-entropy, for binary classification the value C is positive (since all the individual terms in sum is negative) and tends toward 0 as the neuron gets better at computing the desired output y for all training inputs x. Whereas MSE loss function perform better if the output is regression to arbitrary values.

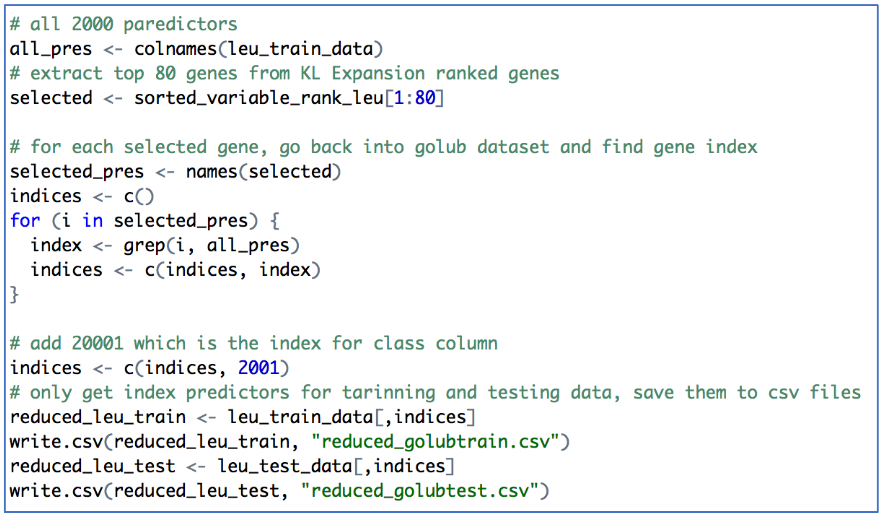
$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)],$$

a: output of neuron
y: desired output
x: training inputs

ii. The optimization function I used is a first order SGD (stochastic gradient descent optimizer). For Golub dataset, I have binary classification and binary cross-entropy function, SGD can minimize or maximize the loss function by finding inflection points from it and affect the accuracy of training data and testing data. Compared to first order optimization, second order optimization function is too costly and does not fit the cross-entropy error function I used since it gives curvature of the error surface.

Question 3e:

Top 80 genes are extracted by using this code and saved as reduced csv file (avoid sourcing kl expansion from last assignment to save time).

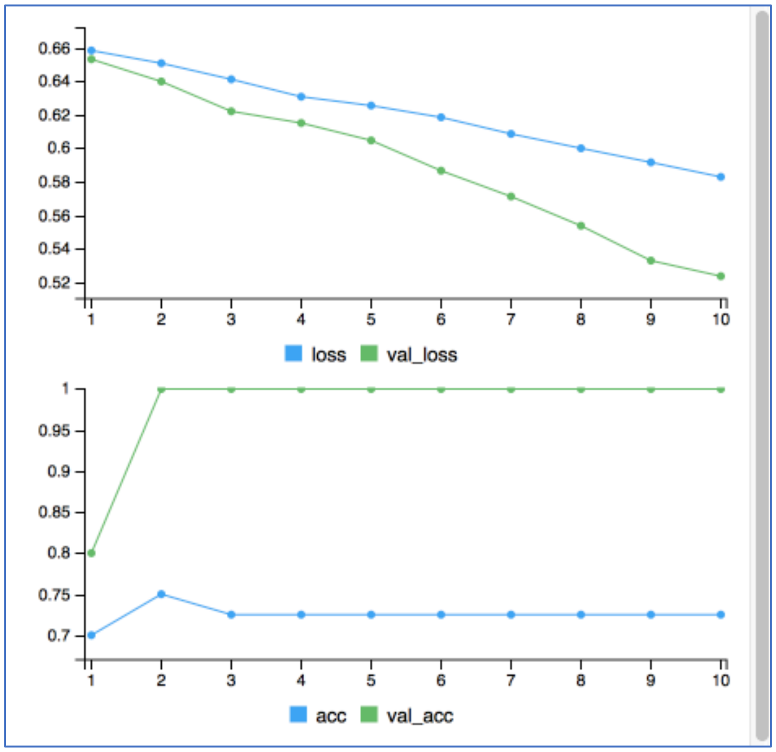


```r
# all 2000 paredictors
all_pres <- colnames(leu_train_data)
# extract top 80 genes from KL Expansion ranked genes
selected <- sorted_variable_rank_leu[1:80]

# for each selected gene, go back into golub dataset and find gene index
selected_pres <- names(selected)
indices <- c()
for (i in selected_pres) {
  index <- grep(i, all_pres)
  indices <- c(indices, index)
}

# add 20001 which is the index for class column
indices <- c(indices, 2001)
# only get index predictors for tarinning and testing data, save them to csv files
reduced_leu_train <- leu_train_data[,indices]
write.csv(reduced_leu_train, "reduced_golubtrain.csv")
reduced_leu_test <- leu_test_data[,indices]
write.csv(reduced_leu_test, "reduced_golubtest.csv")
```

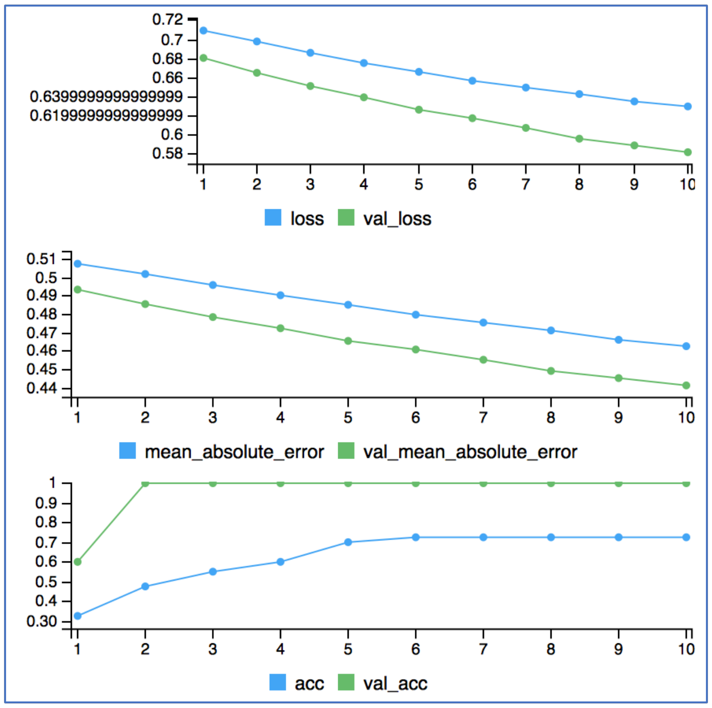
Although within each run the results might be a little different, the final training accuracy is around 72.5%, as testing accuracy is pretty low which is 48%. This suggests that my model is overfitting the data. I set validation_split=0.1, and as the error function shown below, validation set is performing better on both error and accuracy, and this usually is a sign of model not overfitting. However, based on low testing accuracy, the reasons causing this could be small training dataset with large number of input features, the combination of error function and activation function is not optimal, further work need to be done to improve performance of this model.



```
$loss
[1] 0.700294137

$acc
[1] 0.4814814925
```

The three metrics I tried are accuracy, binary_accuracy, c('mae', 'acc'). The result for using ‘accuracy’ and ‘binary_accuracy’ is the same for since I used binary_crossentropy loss function. When using c('mae', 'acc'), I got another output as: mean_absolute_error, which has similar pattern as error function.



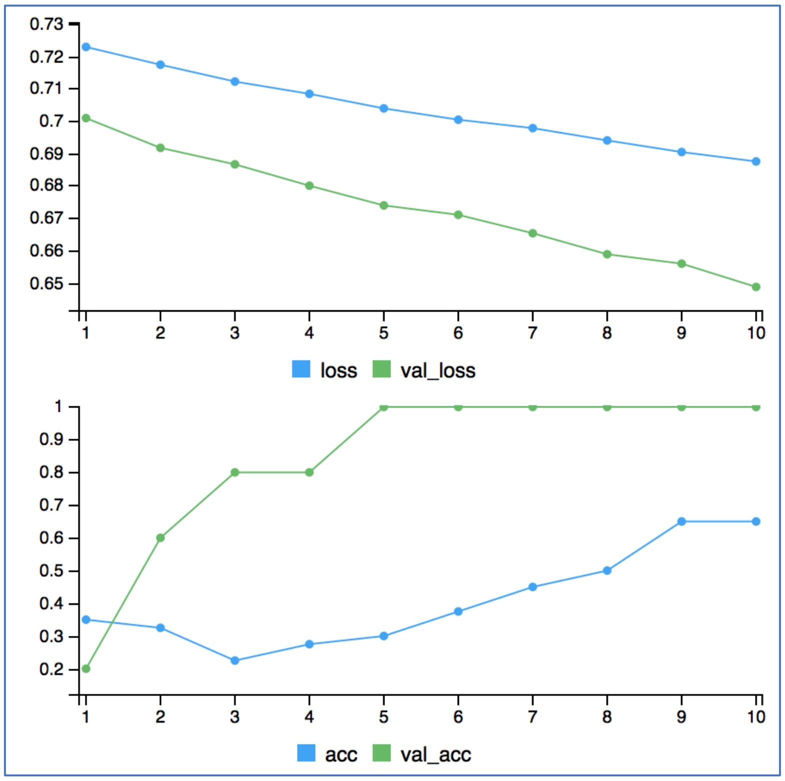
```
$loss
[1] 0.6982257962

$mean_absolute_error
[1] 0.497105062

$acc
[1] 0.4814814925
```

Question 4:

As shown below, the accuracy for this model is lower than previous one. By having fewer neurons going to more neurons, the model is making more classes from previous layer through redrawing boundary, which can decrease accuracy and cause information loss. An ideal neural network should go sequential from more neurons to fewer neurons, in this process weights and errors are adjusted to draw a better boundary for classification.



```
$loss
[1] 0.6966658831

$acc
[1] 0.4444444478
```

Question 5:

Adjusting different parameters and corresponding test accuracy as listed as below, the highest test accuracy is when split ratio is 0.6/0.4, five hidden layers and use sigmoid as final layer activation function.

split ratio (train/test)	hidden layers	accuracy (Sigmoid)	accuracy (tanh)
0.7/0.3	5	0.727	0.727
0.7/0.3	4	0.727	0.727
0.6/0.4	5	0.759	0.724
0.6/0.4	4	0.655	0.724

For Golub dataset, accuracy from multilayer neural network is higher than single layer. When the number of input features are high, multilayer neural network can better adjust weight and error through those hidden layers, however for single neural networks, it is very easy tell that condensing multiple inputs into one output in only one layer can lead to high errors. For small input features, both single layer and multilayer will have similar accuracies. Therefore, whether to use single layer or multilayer neural network depend on the data.

Sources:

https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
http://neuralnetworksanddeeplearning.com/chap3.html