Machine Learning Assignment 4 Hands on with ANN

(40 pts)

Problem1— Using ANN for Classifying News Articles

For this problem, we are going to use reuters dataset. It is a set of short newswires along with their topics published by Reuters in 1986. It is a simple, widely used toy dastaset for text classification. You are going to use a simple feedforward neural network to classify reuters news into 46 different topics/classes.

The reuters dataset comes packaged as part of Keras . Import the dataset from keras using the following commands:

```
Library(keras)
reuters=dataset_reuters(num_words=10000)
```

Take a look at the structure of reuters.dataset. The dataset is already pre-processed and split into train and test sets. Each example in the train or test set represents a news article and is stored as a list of integers where each integer represents the index of a specific word appearing in that article. When you set num_words=1000 in dataset_reuters, then keras keeps only the top 10000 most frequent words in the data. Let's take a look at the structure of the training examples:

```
> str(reuters$train$x)
List of 8982
 $ : int [1:87] 1 2 2 8 43 10 447 5 25 207 ...
 $ : int [1:56] 1 3267 699 3434 2295 56 2 7511 9 56 ...
.............
```

This means that the first article consists of the words with indices 1,2,2,8,etc.
As you can see, the examples in the train and test data are vectors of varying lengths. Neural Networks requires a matrix where each row represents an example and each column represent an attribute of that example. Therefore, to prepare this data for neural networks we need to make sure that all examples have the same number of columns (that is, they are vectors of the same length). To do this, we do one-hot encoding of each example to turn them into vectors of 0s and 1s. This would mean for instance, turning the sequence [1,2,2,8,….] into a 10000 dimensional vector that would be all zeros except for indices 1,2,8, etc. In other words, we turn each news article into a 10000 dimensional binary vector which indicates which of the top 10000 frequent words occur in that article. You can use the following r function to do this one-hot-encoding:

```
one_hot_encoding=function(x, dimension=10000) {
  encoded=matrix(0,length(x),dimension)
  for (i in 1:length(x))
    encoded[i, x[[i]]]=1
  encoded}
```

Where x is the list of examples for which you want to do one-hot-encoding. Call this function on reuters$train$x and reuters$test$x to get the one-hot encoding of the train and test examples. This will produce a matrix with dimensions (8982,10000) for the train data and a matrix with dimensions(2246, 10000) for the test data. You can now use these matrices to train and test your neural network model.

Q1. (5 pts) Create an ANN model to classify reuters news article into 46 classes (note that reuters$train$y and reuters$test$y are vectors of integers between 0-45 representing the category of each news article). Use at least two hidden layers and compute the accuracy of your model on the test set.

Unlike the fashion Mnist dataset where each image had to be flattened into a one dimensional vector, for the reuters dataset, you do not need the flatten layer before the dense layer as each observation (news article) is already one dimensional.

Q2. (5 pts) Split the train data into train/validation set. Use the first 1000 examples in reuters$train for validation and the rest for training your model. Use "tfruns" package to tune your ANN's hyper-parameters including, the number of nodes in each hidden layer, the activation function in each hidden layer, batch_size, learning_rate, and the number of epochs). Validate each model on the validation set. Answer the following questions:

1- Which model (which hyper-parameter combination) resulted in the best accuracy on the validation data? Make sure that you print the returned value from tfruns and report the run with the highest validation accuracy.
2- Print the learning curve of your best model. Does your best model still overfit?
3- Does your validation_loss stop decreasing after several epochs? If so, at roughly which epoch does your validation_loss stop decreasing?

Q3. (5 pts) Now use ALL the training data in reuters$train (i.e., train + validation data) to train an ANN with the best hyper- parameter combination you found after tuning in the previous question. Compute the accuracy of this model on the test set.

Problem2—Predicting Baseball players' salaries

Given the hitters dataset attached with this assignment spec, the goal of this problem is to predict the Salary for baseball players based on other variables in the dataset. The dataset is a collection of Major League Baseball data from the 1986 and 1987 seasons. You can find more information about the dataset here: https://rdrr.io/cran/ISLR/man/Hitters.html

Section1- Exploratory Analysis

1. (1p) Download the dataset hitters.csv and explore the overall structure of the dataset using the str() function. Get a summary statistics of each variable. Answer the following questions:
   - How many observations do you have in the data?
   - How many categorical and numeric variables you have in your data?
   - Is there any missing value?
   - Draw the histogram of salary. Interpret what you see in the histogram.

2. (1p) remove the observation for which Salary value is missing

3. (2 pt) Which predictors have most correlation with Salary? Use scattered plot, side-by-side box plots, t-test and correlation matrix to answer this question.

4. Use set.seed(1) to set the random seed so I can reproduce your results.

5. Use Caret's "createDataPartition" method as follows to partition the dataset into hitters_train, and hitters_test (use 90% for training and 10% for testing)

```
inTrain = createDataPartition(hitters$Salary, p=0.9, list=FALSE)
hitters_train = hitters[inTrain,]
hitters_train = hitters[-inTrain,]
```

The first line creates a random 90%-10% split of data such that the distribution of the target variable hitters$salary is preserved in each split. The `list = FALSE` option avoids returning the data as a list. Instead, inTrain is a vector of indices used to get the training and test data.

6. (1pt) Neural networks do not accept categorical variables and we must encode the categorical variables before training the network. All the categorical variables in this dataset are binary (i.e., have two levels) so you can encode them by simply using iflese function to convert each to a numeric variable with two values 0 and 1.

7. (1pt)Replace the salary column with log(salary) where log is the logarithm function. This will be the attribute we want to predict.

   If you are wondering why we are predicting log(salary) instead of just salary, there are two reasons for it:

   1- Salary variable is right-skewed. A skewed target variable can make a machine learning model biased. For instance, in this case lower salaries are more frequent in the training data compared to the higher salaries. Therefore, a machine learning model trained on this data is less likely to successfully predict higher salaries. When we take the log of a right-skewed distribution, it makes the distribution more symmetrical.

   2- The range of salary is very large causing the gradients of the loss function to also be large. Multiplying a chain of large gradients during backpropagation can result in numeric overflow and you might see a NAN value for loss function after a few epochs of training. This is called exploding gradient problem. By predicting the salary in the log scale we can avoid the exploding gradients problem for this dataset.

8. (1pt) Set.seed(1) and further divide the hitters_train data into 90% training and 10% validation using Caret's "CreateDataPartition" function.

9. (3 pt) Scale the numeric attributes in the training data (except for the outcome variable, Salary). Use the column means and column standard deviations from the training data to scale both the validation and test data (please refer to slide 81, lecture 9). Note: You **should not scale the dummy variables you created in step 6.** You can append the categorical variables to your scaled numeric variables.

(5 pt) Create an ANN model to predict log(salary) from other attributes. Use at least two hidden layers. Use tfruns to tune your model's hyper-parameters including, the number of nodes in each hidden layer, the activation function in each hidden layer, batch_size, learning_rate, and the number of epochs). Validate each model on the validation set. Answer the following questions:

- Print the returned value from tf_runs to see the metrics for each run. Which run (which hyper-parameter combination) gave the best mean squared error on the validation data?
- Print the learning curve for your best model. Does your best model still overfit?
- Does your validation_loss stop decreasing after several epochs? If so, at roughly which epoch does your validation_loss stop decreasing?

Note: The "fit" function in keras does not accept a dataframe and only takes a matrix. If you want to pass a dataframe as training or validation data to the fit function, you must first use **as.matrix** function to convert it to matrix before passing it to the fit function; for example, **as.matrix(your_training_dataframe) or as.matrix(your_validation_dataframe)**

10. (5 pt) Measure the performance of your best model (after tuning) on the test set and compute its RMSE. **Note that you must reverse the log transformation by taking the exp (exponent) of the predictions returned by the neural network model and compare it to the original salary value (without log transformation).** Doing this, helps us get the RMSE in the original scale.

11. (5 pt) Use a simple (or step wise) linear regression model to predict the salary. Train and test your model on the same data you used to train and test your best neural network model. Compare the RMSE of the linear model on the test data with the RMSE of the neural network model. How does your neural network model compare to a simple linear model?

What to Turn in:
You need to create an Rnotebooke consisting of your code and answers to the questions outlined above for problems 1 and 2.

Format of the submission
The submission must be in two formats:

1- **A .html file which contains the preview of your notebook. When you click on preview in R an html file is created in the same directory as your notebook.**
2- **An .rmd file of your notebook**
3- **Any additional R script you used for creating your neural network models.**
4- **The learning curve from your best run. You can just take a screenshot of your best model and submit it with the rest of your files.**