Neural Networks and Deep Learning

Anurag Nagar

CS 6301

Neural Networks

Creating a neural network

We will create a very simple neural network on the dividend dataset.

As we learnt in class, the first step is to load and pre-process and **scale** the data

```
url <- "http://www.utdallas.edu/~axn112530/"
url <- paste(url, "cs6301/data/dividend.csv", sep="")
data <- read.csv(url , header = TRUE)

# Let's create a min-max normalization function:
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
```

Creating a neural network

```
dividend <- as.data.frame(lapply(data, normalize))

# split data into train and test parts
index = sample(1:nrow(dividend),
               round(0.70*nrow(dividend)))
trainset <- as.data.frame(dividend[index,])
testset <- as.data.frame(dividend[-index,])
```

Creating a neural network

```
#Neural Network
library(neuralnet)

nn <- neuralnet(dividend ~ fcfps + earnings_growth + de
                + mcap + current_ratio, data=trainset,
                hidden=c(2,1), linear.output=FALSE,
                threshold=0.01)
```

Analyzing the model output

Below are the important outputs of the model:

1. **nn$result** - the prediction for every data point
2. **nn$weights** - the final fitted weights for the network
3. **nn$generalized.weights** - Generalized weights point the linearity or non-linearity of each variable in predicting the response variable. The distribution of the generalized weights points out whether a variable has a linear, non-linear or no effect in predicting the output. If the distribution has low variance, it indicates a linear effect, a high distribution indicates a non-linear effect, and if all values are close to 0, it indicates no effect.

Analyzing the model output

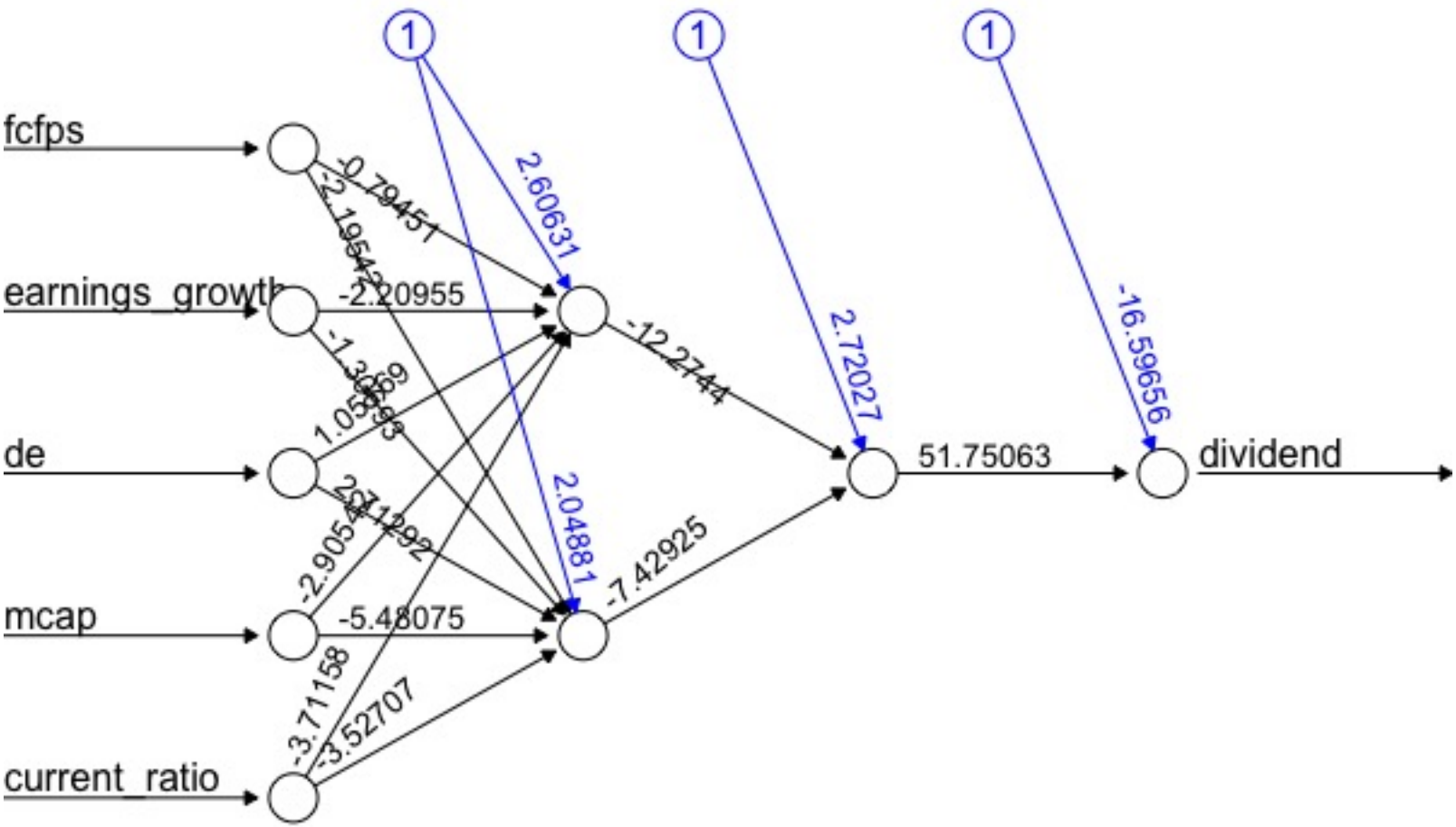
4. **nn$data** contain the normalized input value of the entire dataset
5. **nn$response** contains the normalized input value for the output or response variable.
6. **nn$covariate** contains the normalized input value for all the independent variable (also called covariates)

Analyzing the model output

7. **nn$result.matrix** contains the final weights in a matrix format
8. **plot(nn)** is a visual representation of the final weights

```
plot(nn)
```

Analyzing the model output



Error: 1.005096 Steps: 581

References

1. Interpreting neural network results: https://pdfs.semanticscholar.org/51f8/f33fc707e6c8da2c1562b8ead4f7bae629a6.pdf
2. R journal article on NN: https://journal.r-project.org/archive/2010/RJ-2010-006/RJ-2010-006.pdf
3. Neural Net: Further Insights: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5009026/
4. Code from: https://www.r-bloggers.com/neuralnet-train-and-test-neural-networks-using-r/

Moving towards Deep Learning

Getting started

For deep learning, we will be using the following R packages:

- tensorflow
- keras
- cloudml

Suggested installation method

```
devtools::install_github("rstudio/keras")
library(keras)
install_keras()
```

Getting started with Google cloud

To proceed, you will need to create an account on Google cloud: https://cloud.google.com

You will need to enter a credit card, after which you will be given $300 credit. You can choose the option not to autocharge you.

It's completely your responsibility to manage your account. The instructor or department will not be liable for any financial loss.

Google cloud call from R

Run following commands to install Google cloud.

```
gcloud_install()
```

Starting Google cloud server

After creating a project, you should start the **Compute Engine** within your Google account

From R, type the following command

```
gcloud_init()
```

To submit a job:

```
library(cloudml)
cloudml_train("Complete Path to your R file")
```

It will take much longer for the first job to run, as all packages need to installed on your engine

Submitting a Keras job

As a starter, we will run the code from this link:

https://tensorflow.rstudio.com/tools/cloudml/articles/getting_started.html

Training a Deep Network

Understanding the Sequential Model

Sequential model consists of layers added on top of each other starting from the input to the output layer.

```
library(keras)
model <- keras_model_sequential()
model %>%
  layer_dense(units = 32, input_shape = c(784)) %>%
  layer_activation('relu') %>%
  layer_dense(units = 10) %>%
  layer_activation('softmax')
```

Understanding the Sequential Model

The first hidden layer is dense of 32 units and has 784 inputs.

```
> summary(model)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
activation_1 (Activation)    (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
```

The last layer is softmax which gives for each data point, the probability of each of the classes.

How to figure out number of parameters

Dense layer 1 = 784 * 32 connections + 32 bias units (one for each neuron)

= 25120

Dense layer 2 = 32 * 10 + 10

= 330

Note that the activation layer simply transforms the input. In case of ReLu no parameters are needed.

Model Compilation

Model compilation requires a loss function and an optimizer algorithm

```
model %>% compile(
  optimizer = 'rmsprop',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)
```

```
model %>% compile(
  optimizer = optimizer_rmsprop(lr = 0.002),
  loss = 'mse'
)
```

Model Fitting

Model can now be fitted on a dataset. The following is based on a dummy dataset

```
# Generate dummy data
data <- matrix(runif(1000*100), nrow = 1000, ncol = 100)
labels <- matrix(round(runif(1000, min = 0, max = 9)),
                 nrow = 1000, ncol = 1)

# Convert labels to categorical one-hot encoding
one_hot_labels <- to_categorical(labels,
                                 num_classes = 10)

# Train the model, iterating on the data in batches of 32 s
model %>% fit(data, one_hot_labels,
              epochs=10, batch_size=32)
```

Practice Question

Deep Learning for Text Classification

```
library(keras)
imdb <- dataset_imdb(num_words = 10000)
c(c(train_data, train_labels),
  c(test_data, test_labels)) %<-% imdb
```

Note that this is pre-processed data, with stop words removed and words convered to vectors.

We are limiting ourselves to top 10,000 most frequent words.

Deep Learning for Text Classification

Next we need to ensure that all sequences have same length so we can feed them into the neural net

```
vectorize_sequences <- function(sequences,
                                dimension = 10000) {
  # Create an all-zero matrix of shape
  #       (len(sequences), dimension)
  results <- matrix(0, nrow = length(sequences),
                    ncol = dimension)
  for (i in 1:length(sequences))
    # Sets specific indices of results[i] to 1s
    results[i, sequences[[i]]] <- 1
  results
}
```

Deep Learning for Text Classification

```
# Our vectorized training data
x_train <- vectorize_sequences(train_data)
# Our vectorized test data
x_test <- vectorize_sequences(test_data)

# Vectorize Labels
y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)
```

Building the Network

We will create a network of 3 layers having 16, 16 and 1 neurons. The last one is the output layer.

```
library(keras)
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu",
              input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

Defining loss function and other parameters

```
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)
model %>% fit(x_train, y_train, epochs = 4,
              batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
```

Looking at the results

Let's look at the results.

```
results
```

What could you have done to make the results better?

Another approach

This time we will split data into train and validation parts

```
val_indices <- 1:10000
x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

Another approach

```
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
```

Assignment

Text Classification Problem

Steps

1. You have to identify a text classification problem and dataset from UCI text datasets

You are responsible for splitting the data into two parts: train and test

2. You have to preprocess the data. It involves removing stop words, converting words to vectors, choosing how many words you would like to use and finally make the data suitable for being used as an input layer.

Hint: It's totally upto you how you do this. A good resource is the *tm package* in R

Text Classification Problem

Steps

3. Create a deep network using Keras package. It's upto you to design this. Remember to design the output layer based on the data features.
4. **You need to run your code on Google Cloud with GPU enabled. We had given you an introduction in class. You will need to figure out the details - such as how to transfer to cloud and refer to them in your code**

Some useful sites:

https://cloud.google.com/storage/docs/creating-buckets

gs_copy function of cloudml package
https://cran.r-project.org/web/packages/cloudml/cloudml.pdf

Text Classification Problem

Steps

5. Split the **train** data into train and validation parts. Use the fit function to build and validate your data and create a plot of the history.
6. Apply the model on the test part and see how well it does.
7. How can you improve the performance. You need to vary the parameters and run your code at least 5 times and report the parameters used and accuracy obtained.