Module 11: Recommended Exercises

TMA4268 Statistical Learning V2020

Stefanie Muff/Martina Hall/Michail Spitieris, Department of Mathematical Sciences, NTNU

March xx, 2020

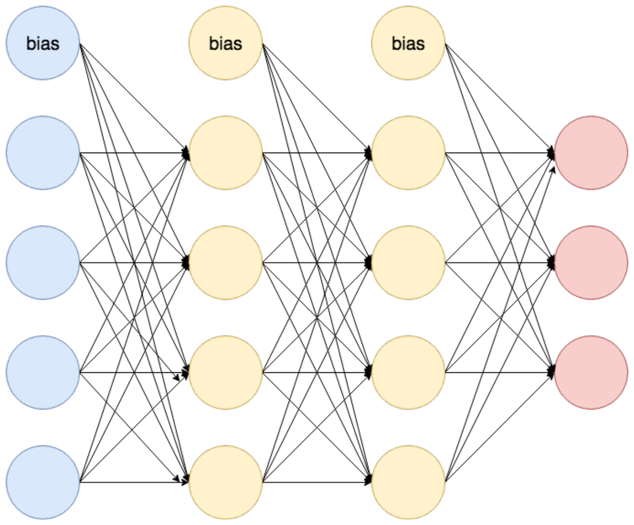
Contents

Theoretical exercises

Problem 1

a)

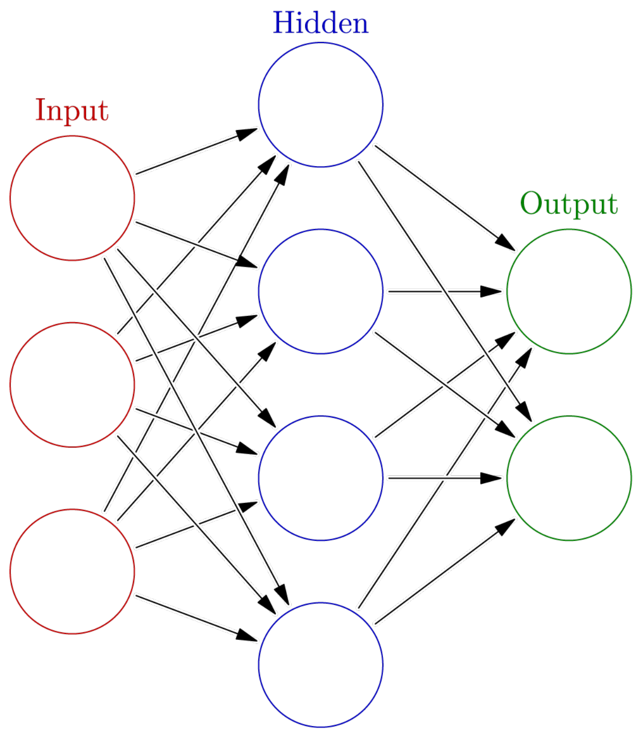
Write down the equation describing this network. What would you call such a network?



b)

The following image is the illustration of an artificial neural network at Wikipedia.

- What can you say about this network architecture
- What do you think it can be used for (regression/classification)?



https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg

c)

Given a the following problems, what are sensible feedforward network architectures (depth, width, activation function) and methods (loss function, algorithms) that you would explore?

- Regression with on univariate response, 10 possible covariates, 500 observations.
- Classification with two classes, 100 possible covariates, 10000 observations.
- Classification with 10 classes, image data (like the MNIST), 50000 observations.

d)

What are the similarities and differences beween a feedforward neural network with one hidden layer with `linear` activation and `sigmoid` output (one output) and logistic regression?

e)

In a feedforward neural network you may have 10000 weights to estimate but only 1000 observations. How is this possible?

f)

Which network architecture and activation functions does this formula give?

$$\hat{y}_1(\mathbf{x}) = \beta_{01} + \sum_{m=1}^{5} \beta_{m1} \cdot \max(\alpha_{0m} + \sum_{j=1}^{10} \alpha_{jm} x_j, 0)$$

How many parameters are estimated in this network?

g)

Which network architecture and activation functions does this formula give?

$$\hat{y}_1(\mathbf{x}) = (1 + \exp(-\beta_{01} - \sum_{m=1}^{5} \beta_{m1} \max(\gamma_{0m} + \sum_{l=1}^{10} \gamma_{lm} \max(\sum_{j=1}^{4} \alpha_{jl} x_j, 0), 0))^{-1}$$

How many parameters are estimated in this network?

h)

In a regression setting: Consider

- a sum of non-linear functions of each covariate in Module 7
- a sum of many non-linear functions of sums of covariates in feedforward neural networks (one hidden layer, non-linear activation in hidden layer) in Module 11.

Explain how these two ways of thinking differ? Pros and cons?

i)

What is the most interesting aspect of neural network (your opinion)? How would you compare how feedforward neural networks are fitted as compared to fitting multiple linear regression and logistic regression? Compare how model selection (which covariates) are performed.

Practical exercises

You should be able to

- Fit regression
- Fit classification with two classes
- Fit classifiation with more than two classes

all with the `nnet` package. Below is an example for classifiation with more than two classes, and a small part with `keras` is added for completeness.

Problem 2: Handwritten digit recognition data

See Friedman, Hastie, and Tibshirani (2001) Section 11.7.

Description:

(direct quote from `?zip.train` in `ElemStatLearn` R package)

Normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service. The original scanned digits are binary and of different sizes and orientations; the images here have been deslanted and size normalized, resulting in 16 x 16 grayscale images.

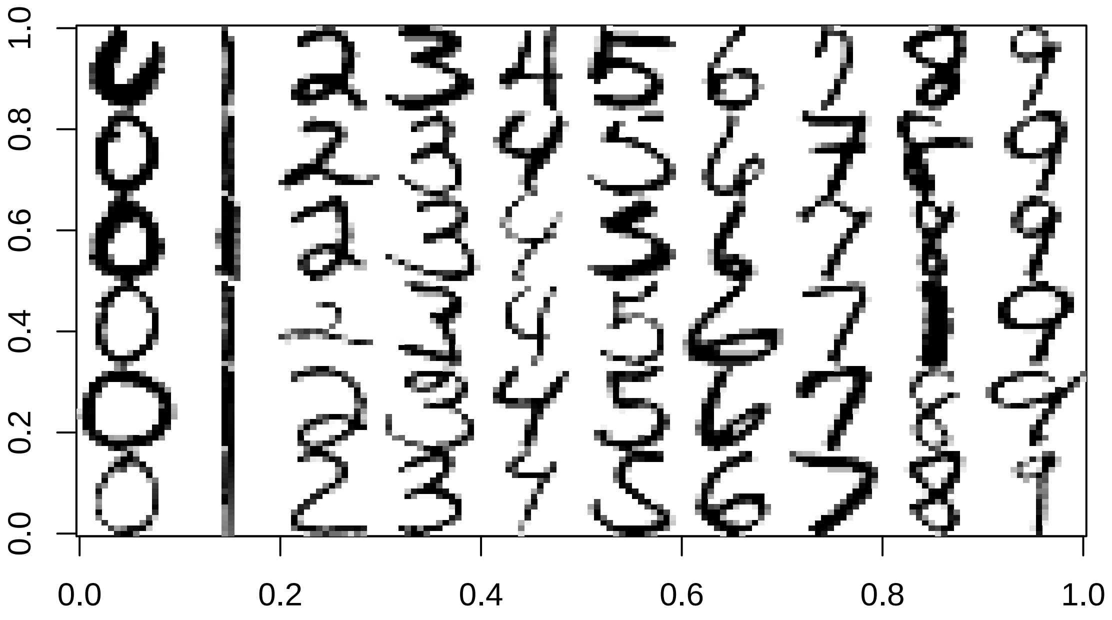
The data are in two gzipped files, and each line consists of the digit id (0-9) followed by the 256 grayscale values.

There are 7291 training observations and 2007 test observations.

The test set is notoriously "difficult", and a 2.5 excellent. These data were kindly made available by the neural network group at AT&T research labs (thanks to Yann Le Cunn).

A selection of some images

```r
library(ElemStatLearn)
# code from help(zip.train)
findRows <- function(zip, n) {
    # Find n (random) rows with zip representing 0,1,2,...,9
    res <- vector(length = 10, mode = "list")
    names(res) <- 0:9
    ind <- zip[, 1]
    for (j in 0:9) {
        res[[j + 1]] <- sample(which(ind == j), n)
    }
    return(res)
}
digits <- vector(length = 10, mode = "list")
names(digits) <- 0:9
rows <- findRows(zip.train, 6)
for (j in 0:9) {
    digits[[j + 1]] <- do.call("cbind", lapply(as.list(rows[[j + 1]]),
        function(x) zip2image(zip.train, x)))
}
im <- do.call("rbind", digits)
image(im, col = gray(256:0/256), zlim = c(0, 1), xlab = "", ylab = "")
```



Preprocessing data - scaling

Important to decide on the scale based on the training data, and then apply to both training and test data.

```
train_data = zip.train[, -1]
train_labels = factor(zip.train[, 1])
test_data = zip.test[, -1]
test_labels = factor(zip.test[, 1])
mean <- apply(train_data, 2, mean)
std <- apply(train_data, 2, sd)
train_data <- scale(train_data, center = mean, scale = std)
test_data <- scale(test_data, center = mean, scale = std)
```

Check out 5 hidden nodes

5 hidden nodes: $257 * 5 + 6 * 10$=257*5+6*10 parameters

```
library(nnet)
zipnnet5 <- nnet(train_labels ~ ., data = train_data, size = 5, MaxNWts = 3000,
    maxit = 5000)
summary(zipnnet5)
pred = predict(zipnnet5, newdata = test_data, type = "class")
library(caret)
confusionMatrix(factor(pred), test_labels)
```

The above took some time to run, the results were:

```
> zipnnet5<- nnet(train_labels~., data=train_data,size=5,MaxNWts=3000,maxit=5000)
iter2960 value 864.566658
final  value 864.561810
converged
> summary(zipnnet5)
a 256-5-10 network with 1345 weights
options were - softmax modelling
   b->h1    i1->h1    i2->h1    i3->h1    i4->h1    i5->h1    i6->h1    i7->h1    i8->h1    i9->h1   i10->h1   i11-
  -49.27      9.15      1.24     21.03     -2.82     17.97      4.63     11.60     -4.31      2.28     -4.57     -
 i19->h1   i20->h1   i21->h1   i22->h1   i23->h1   i24->h1   i25->h1   i26->h1   i27->h1   i28->h1   i29->h1   i30-
> confusionMatrix(factor(pred),test_labels)
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2   3   4   5   6   7   8   9
         0 324   0   6   5   3   9   7   0   3   0
         1   1 245   7   0   1   0   0   0   7   5
         2   4   7 148   8  12   5  11   0   3   0
         3   2   0   6 128   4  10   0   4   5   3
         4   4   1   4   0 152   1   1   9   2   4
         5   1   0   2  18   1 117   5   0   7   1
         6  21   3  11   0   6   1 146   0   3   0
         7   0   1   6   4   5   1   0 122   9   5
         8   2   2   7   3   6  15   0   1 113   3
         9   0   5   1   0  10   1   0  11  14 156
```

```
Overall Statistics

               Accuracy : 0.8226
                 95% CI : (0.8052, 0.8391)
```

Slightly better and faster with keras, but we see that we probably need more hidden layers since this is a difficult problem! An introduction to `keras` is given later in this module.

```
library(keras)
network <- keras_model_sequential() %>% layer_dense(units = 5, activation = "sigmoid",
    input_shape = c(16 * 16)) %>% layer_dense(units = 10, activation = "softmax")
network %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
train_data <- array_reshape(train_data, c(7291, 16 * 16))
train_data <- train_data/255
train_labels <- to_categorical(train_labels)

test_data <- array_reshape(test_data, c(2007, 16 * 16))
test_data <- test_data/255
org_test_labels <- test_labels
test_labels <- to_categorical(test_labels)

fitted <- network %>% fit(train_data, train_labels, epochs = 500, batch_size = 128,
    validation_split = 0.2)
library(ggplot2)
plot(fitted) + ggtitle("Fitted model")  #same as above

network %>% evaluate(test_data, test_labels)
# acc : num 0.873
library(caret)
res <- network %>% predict_classes(test_data)
confusionMatrix(factor(as.character(res)), org_test_labels)
```

10 hidden nodes

10 hidden nodes: $257 * 10 + 11 * 10$=257*10+11*10 parameters

```
library(ElemStatLearn)
train_data = zip.train[, -1]
train_labels = factor(zip.train[, 1])
test_data = zip.test[, -1]
test_labels = factor(zip.test[, 1])
mean <- apply(train_data, 2, mean)
std <- apply(train_data, 2, sd)
train_data <- scale(train_data, center = mean, scale = std)
test_data <- scale(test_data, center = mean, scale = std)
library(keras)
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "sigmoid",
    input_shape = c(16 * 16)) %>% layer_dense(units = 10, activation = "softmax")
network %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
train_data <- array_reshape(train_data, c(7291, 16 * 16))
train_data <- train_data/255
train_labels <- to_categorical(train_labels)
```

```
test_data <- array_reshape(test_data, c(2007, 16 * 16))
test_data <- test_data/255
org_test_labels <- test_labels
test_labels <- to_categorical(test_labels)

fitted <- network %>% fit(train_data, train_labels, epochs = 500, batch_size = 128,
    validation_split = 0.2)
allfitted <- network %>% fit(train_data, train_labels, epochs = 500,
    batch_size = 128)
network %>% evaluate(test_data, test_labels)
# acc : num 0.9133034
```

Problem 2: Classification of diabetes cases (Compulsory problem from 2018)

During our lectures, we have used a densely connected neural network to classify a movie review as either positive or negative using the IMDB data set. We will use the same data here, refer to the lecture slides for reading the data set and setting up the network.

More specifically, we have defined a linear stack of dense layers, containing two intermediate layers containing 16 hidden units each with reLu activation functions.

```
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")
```

Q20. What is the advantage of using a non-linear activation function such as relu?

Q21. Why do we need to use a different activation function (sigmoid) in the output layer instead of using relu again?

In one of our lectures, we have fitted the model above and plotted training and validation loss as well as accuracy, as illustrated in the figure below.

Keep two intermediate layers in the network. Try creating a simpler model with 4 hidden units instead of 16 and a more complex model with 32 hidden units instead of 16.

Q22. Plot the training and validation loss and accuracy for the simpler and more complex model mentioned above. How do they compare with the model with 16 hidden units?

Q23. Besides reducing the network's size, what other methods can be used to avoid overfitting with neural network models? Briefly describe the intuition behind each one.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer series in statistics New York.