

# Module 11: Recommended Exercises

Statistical Learning V2021

alexaoh

24 april, 2021

## Problem 1

a)

Write down the equation that describes and input is related to output in this network, using general activation functions  $\phi_o$ ,  $\phi_h$  and  $\phi_{h^*}$  and bias nodes in all layers. What would you call such a network?

The given network is called a 4-4-4-3 feedforward network, with one bias node in each layer (except the output of course). The equation that describes the network can be written as shown in the following. The nodes in the first hidden layer are denoted by

$$z_m(x) = \phi_{h^*}(\alpha_{0m} + \sum_{j=1}^4 \alpha_{jm} x_j), \quad m = 1, \dots, 4,$$

where  $\alpha_{jm}$  is the weight from input  $j$  to hidden node  $m$  and  $\alpha_{0m}$  is the bias term for the  $m^{th}$  hidden node. The nodes in the second hidden layer can be denoted in the similarly, e.g. with the letter  $y_l(x)$ ,  $l = 1, \dots, 4$ , with weights  $\gamma_{ij}$  and  $l = 1, 2, 3, 4$  nodes. Finally, the output layer can be denoted by

$$y_c(\mathbf{x}) = \phi_o(\beta_{0c} + \sum_{l=1}^4 \beta_{lc} y_l) = \phi_o(\beta_{0c} + \sum_{l=1}^4 \beta_{lc} \phi_h(\gamma_{0l} + \sum_{m=1}^4 \alpha_{ml} z_m)) = \phi_o(\beta_{0c} + \sum_{l=1}^4 \beta_{lc} \phi_h(\gamma_{0l} + \sum_{m=1}^4 \alpha_{ml} \phi_{h^*}(\alpha_{0m} + \sum_{j=1}^4 \alpha_{jm} x_j))),$$

for  $c = 1, 2, 3$ , i.e. all the output nodes.

b)

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

- What can you say about this network architecture? *Answer:* It is feedforward. Moreover, the hidden layer has four nodes, but no bias node, since there are weights “coming into” each of the nodes. The input layer has either 3 input nodes or 2 input nodes and a bias node. The output layer has two nodes.
- What do you think it can be used for (regression/classification)? *Answer:* It can be used for binary or tertiary classification (depending on how the nodes are defined), since there are two nodes in the output layer. Softmax activation function can be used for two classes. However, for two classes, one output node is most commonly used. Moreover, it can be used for regression with two responses.

c)

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

- Similarities: The neural network gives approximately the same coefficients as the logistic regression, since the input is “squashed” into the interval  $[0, 1]$  when output from the network, because of the sigmoid activation function. They will not give the exact same coefficient estimates, since they use different algorithms to reach minimum.
- Differences: Use different algorithms. The neural network must estimate a lot more coefficients. The logistic regression is much better for interpretation compared to the neural network. However, the hidden layer may find latent structures in the data, with the help of the hidden layer, despite the activation function of the hidden layer being linear.

d)

In a feedforward neural network you may have 10'000 weights to estimate but only 1000 observations. How is this possible?

This is possible because you may have 1000 observations in the input layer, but an arbitrarily deep neural network, with arbitrarily wide layers. This is possible to calculate because of the use of iterative methods (gradient descent and backpropagation), which means that there are no unique solutions.

## Problem 2

a)

Which network architecture and activation functions does this formula correspond to?

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

The hidden layer has the ReLu activation function. Moreover, the network has 10 input nodes (in addition to a bias node) and one hidden layer, with 5 nodes (in addition to a bias node).

How many parameters are estimated in this network?

Hence, in this network,  $11 \cdot 5 + 6 = 61$  parameters are estimated.

b)

Which network architecture and activation functions does this formula give?

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

This corresponds to a network with 4 input nodes, one hidden layer with 10 nodes (in addition to a bias node), another hidden layer with 5 nodes (in addition to a bias node) and one output node. The input layer and the first hidden layer use the ReLu activation function, while the second hidden layer uses a sigmoid function.

How many parameters are estimated in this network?

Hence, in this network,  $4 * 10 * 5 + 1 = 201(?)$  parameters are estimated.

c)

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?

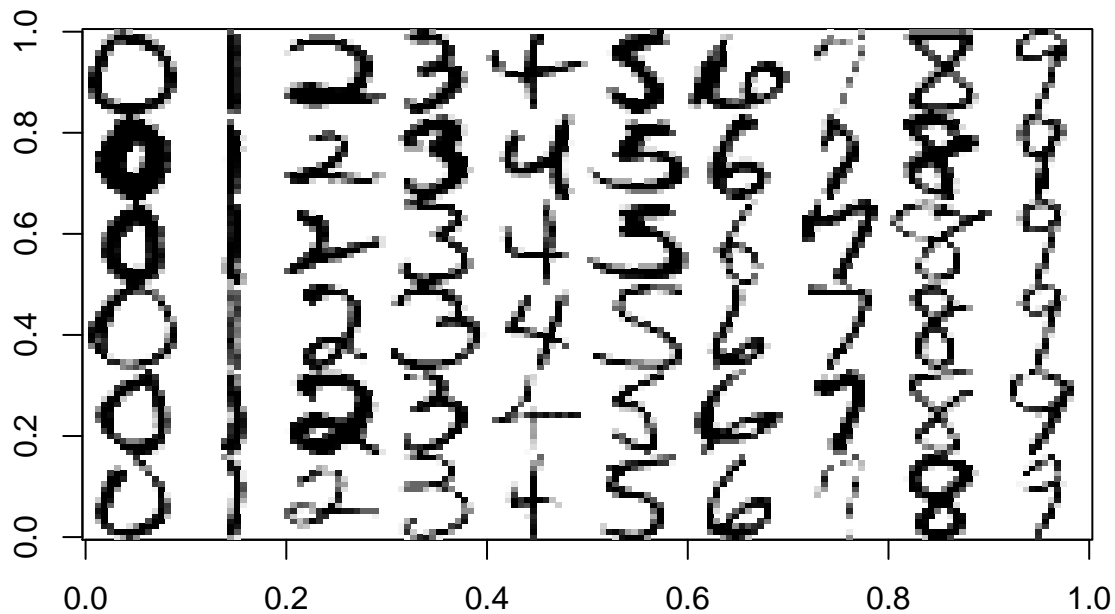
The first model is simpler, while the second is more complex. Pros and cons?

### Problem 3: Handwritten digit recognition data

The following problem involves classification of handwritten digits from 0 to 9. For more details see `?zip.train` in `ElemStatLearn` R package or see Section 11.7 in the Book Elements of Statistical Learning.

Note: the `ElemStatLearn` package has been archived from CRAN, so to get ahold of the data, download the archived package from [https://cran.r-project.org/src/contrib/Archive/ElemStatLearn/ElemStatLearn\\_2015.6.26.tar.gz](https://cran.r-project.org/src/contrib/Archive/ElemStatLearn/ElemStatLearn_2015.6.26.tar.gz), and then in RStudio, click the “Packages” tab, then click “Install”, select “Install from: Package Archive File”, and then select the `.tar.gz` file you downloaded above.

We have images of handwritten digits in grayscale of size  $16 \times 16$  which are stored as vectors of size 256. There are 7291 training observations and 2007 test observations



a)

Preprocessing is important before we fit a neural network. Standardize the features and transform the digits to factors for both train and test data.

Remember that it is important to decide on the scale based on the training data, and then apply the same standardization for both training and test data (i.e., centering around the same mean and transforming to the same variance).

b)

Use a feedforward-network with one hidden layer to train and predict using the `nnet()` function from the `nnet` R package. Include 5 hidden nodes plus a bias node in the input and hidden layers. What is the number of parameters? Look at the confusion table for the test data. How good does the network perform?

## Problem 4: Deep Learning with Keras

We again work on a handwritten digit recognition problem, but now we use the data from the keras library. This dataset consists of 60000 training data of  $28 \times 28$  grayscale images and 10000 test data.

For more info see [here](#)

If you have already installed `keras` ignore the following chunk of code

```
# Install the keras R package install.packages('keras') Install the
# core Keras library + TensorFlow
library(keras)
install_keras()
# for machines with NVIDIA GPU
install_keras(tensorflow = "gpu")
```

### Data Preprocessing

```
library(keras)
mnist = dataset_mnist()
x_train = mnist$train$x
y_train = mnist$train$y
x_test = mnist$test$x
y_test = mnist$test$y
# reshape
x_train = array_reshape(x_train, c(nrow(x_train), 28 * 28))
x_test = array_reshape(x_test, c(nrow(x_test), 28 * 28))
# rescale
x_train = x_train/255
x_test = x_test/255
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

a)

In order to fit a model in keras we can use the following steps. We fit a densely (or fully) connected neural network with 2 hidden layers. Fill in the missing inputs and run the model.

```
model = keras_model_sequential() %>%
  layer_dense(units = 8, activation = 'relu', input_shape = c(...)) %>% # fill in the length of the inp
  layer_dense(units = 8, activation = 'relu') %>%
  layer_dense(units = 10, activation = ...) # fill in the name of the activation function
```

**1. Define the model** Possible choices of the activation function are 'sigmoid', 'softmax' or the identity function (which means that we don't have to define any activation argument in the last layer). What is the use of identity function?

```
model %>% compile(optimizer = "rmsprop", loss = ..., metrics = c(...))
```

**2. Compile** Possible choices of loss function are 'binary\_crossentropy', 'categorical\_crossentropy' and 'mse'. For metrics you can use 'mean\_absolute\_error' or 'accuracy'.

```
history = model %>% fit(x_train, y_train, epochs = 20, batch_size = 128,  
  validation_split = 0.2)
```

**3. Train** You can find more information about the different arguments in the `fit()` function here: <https://keras.rstudio.com/reference/fit.html>

Report the accuracy on the test data.

```
str(history)  
plot(history)  
model %>% evaluate(x_test, y_test)
```

What is the number of parameters?

b)

Now fit a model with 2 hidden layers with 128 units each. What do you see?

c)

In order to avoid the problem we have seen in b), we can use weight regularization (L1 and L2 norms) or dropout. Apply these methods to the network from b).

**R-hint:** for weight regularization you can use

```
layer_dense(units = 16, activation = "relu", input_shape = 28*28,  
  kernel_regularizer = regularizer_l2(l = 0.001))
```

and for dropout you can use `layer_dropout(0.6) %>%` after each hidden layer. For more details see here.