Introduction to R

Andreas Masuhr

Manuel Stapper

Summer Term 2019

Machine Learning

In this exercise you'll learn to use a neural network to recognize handwritten digits. First, let's get familiar with the data set:

1. `load` the dataset `digits.RData`. It contains all digits that you wrote down, before. All digits have been augmented by rotating and shifting by different angles and directions. This (artificially) increases the sample size and helps training. The data have already been split up into a training (`data.train`) and a test (`data.test`) data set. The images are scaled down to 35×35 pixel and were converted to gray scale (white: 0; black: 1).

2. Use the `source()` command to load all functions from `NeuralNetwork_exercise.R`. A list of the functions you'll need to solve the exercises is given in the appendix of this sheet.

3. How do your hand writings look like? In order to look at the kth test digit you wrote down, plot row $(ID - 1) * 20 + k$ from `data.test$images` using the `digitplot()` function.

4. To train the net, we need an activation function for the neurons. There already is a prototype `net.sigmoid()`, but it is missing the actual computation. In the R code, read the comments above the function `net.sigmoid()` and complete it:

$$f(z) = \frac{1}{1 + e^{-z}}$$
$$z = \boldsymbol{W}x + b$$

   Create vectors $w = (1, 2, 3)$, $b = -3$ and $data = (0.5, 0.5, 0.5)$ and compute `net.sigmoid(w, b, data)`. Your result should be `0.5`

5. Now, create the neural network that you're going to train and save it in the object `digitnet`. The function `net.create()` initializes a neural net given a vector of the sizes of the layers of the network as inputs. Since the images are scaled down to 35×35 pixel we need an input layer of $35^2 = 1225$ neurons and because there are ten different digits, we'll be using an output layer of ten neurons. For a first attempt, we will use a single hidden layer of 30

neurons. Create a vector `sizes` that contains the sizes of input, hidden and output layer and use `net.create(sizes)` to initialize the network and save it in the object `digitnet`.

6. Your network is currently not trained. Use `net.evaluate()` to assess the precision of your net, based on the test data. How many digits are recognized correctly?

7. Now, train the network. Therefore, use the function `net.SGD()`. The function requires the following arguments:

   `net` A (untrained) neural network: `digitnet`

   `training.data` A list with training data. The first entry is a matrix of images, the second a vector of true values: `data.train`

   `epochs` The number of training epochs: `5`

   `mini.batch.size` How many data points are used in each mini batch? `10`

   `eta` The learning rate paramter: `1`

   `test.data` The test data set; same data type as training.data: `data.test`

   `cf` The type of cost function: `"squares"`.

   To save time, we only set the number of training epoch to 5. How well does the net perform?

8. There are many potential reasons (training is too short, choice of activation function, choice of cost function, initialization of the network, sample is too small, overfitting, etc.) why the net performs this bad and we'll look into two of them now: The size of the net, and the choice of the cost function. Create another neural network, `digitnet.large` that contains a hidden layer of 100 neurons. Train this network with the same set up as before. What happenes? Now, change `cf = "ce"` to use the cross entropy cost function and set the learning rate to $\eta = 0.5$. How does the network perform, now?

9. Use `net.IO.plot()` to asses the (trained) net for some of your own handwritings.

10. Use the 20 numbers of your own test data set to evaluate the overall precision for your handwriting. Create a list that contains the images of your 20 digits as first element and a vector with the true values as second element and use this list in `net.evaluate()`.

A Neural network - Objects and functions

- NeuralNetwork: A list with five entries. The first is a vector containing the size of each layer, the second is a list of matrices containing all weights, the third is a list of matrices containing all biases. Elements four and five are dummies for the gradient and will not be needed in the exercises.
- net.create(sizes):
  - Input:
    - sizes: numeric vector. Length equals number of layers. Each entry equals number of neurons in the respective layer.
  - Output: Returns a neural network. Consists of a list with two elements. The first is a list with weights, the second is a list with biases.
- net.IO(net, data, full):
  - Inputs:
    - net: A neural network
    - data: a vector containing an image of a digit
    - full: boolean; should all intermediate output be returned?
  - Output:
    - full = F: a numeric vector; predictions for each neuron of the output layer
    - full =T: a list with numeric vectors that contain the predictions in each layer
- net.SGD(net, training.data, epochs, mini.batch.size, eta, lambda, test.data, cf, ISP, ISP.interval)
  - Inputs:
    - net: A neural network
    - training.data: a list of training data. First entry is a matrix with training images of dimension $n_{Images} \times (n_{pixel;row} \cdot n_{pixel;column})$; second entry is a vector with the true values of the digits of length n_{Images}.
    - epochs: The number of training epochs.
    - The number of digits that is used in each gradient approximation.
    - eta: The learning rate.
    - lambda: The regularization rate; not necessary for the exercises.

    * test.data: a test data set. If included, the network will be evaluated using `net.evaluate()` at the end of each epoch and the best performing net will be returned. Structure is the same as training.data.
    * ISP: boolean; should the in-sample-performance be evaluated? Useful to detect overfitting; not necessary for the exercises.
    * ISP.interval: int; interval (measured in epochs) for in-sample-performance evaluation; not necessary for the exercises.
  - Output: The trained neural network. If test data is supplied, the best performing net will be returned, otherwise the network after the last epoch of training.

- net.evaluate(net, test.data)
  - Inputs:
    * net: a neural network
    * test.data: a list of test data. First entry is a matrix with test images of dimension $n_{testimages} \times (n_{pixel;row} \cdot n_{pixel;column})$; second entry is a vector with the true values of the test digits of length $n_{testimages}$.

- net.sigmoid(weights, biases, data):
  - Inputs:
    * weights: matrix or vector containing weights of a single neuron or a whole layer of neurons.
    * biases: vector or scalar containing biases of a single neuron or a whole layer of neurons.
    * data: vector or scalar contains the inputs for the neurons; either image data or output of neurons of previous layers.
  - Output: returns the fraction of correctly predicted digits.
  - Output: Vector or scalar; prediction of a single neuron or a whole layer of neurons.

- net.sigmoid.prime(weights, biases, data)
  - Inputs:
    * weights: matrix or vector containing weights of a single neuron or a whole layer of neurons.
    * biases: vector or scalar containing biases of a single neuron or a whole layer of neurons.

        * data: vector or scalar contains the inputs for the neurons; either image data or output of neurons of previous layers.
    - Output:
        * Computes the derivative of the sigmoid activation function wrt $z = \boldsymbol{W}'x + b$

- digitplot(digit, nrow, ncol, trueval, predictval, ...):
    - Inputs:
        * digit: Image data for a single digit.
        * nrow, ncol: number of rows and columns of the image. Standard is $\sqrt{length(digit)}$
        * trueval: True digit; standard is `NA`
        * predictval: Predicted value; standard is `NA`
    - Output: No value returned; Plots a handwritten digit in as gray scale image. trueval and predictval are optional and will be added to `main()`.

- net.IO.plot(net, data, trueval):
    - Inputs:
        * net: A neural network.
        * data: Vector containing image data of a single handwritten digit.
        * trueval: Numeric containing the true digit.
    - Output: No value returned; Plots both, the handwritten digit and input/output of the neural network.