Computer Lab 2 - Deep learning and Gaussian processes

Machine Learning 7.5 credits

Mattias Villani, Department of Statistics, Stockholm University

INSTRUCTIONS:

- The sections named Intro do **not** have any problems for you. Those sections contain code used to set up the data and do some initial analysis, so just read and follow along by running each code chunk.
- Your problems are clearly marked out as Problem 1, Problem 2 etc. You should answer all problems by adding code chunks and text below each question.
- Your submission in Athena should contain two files:
  - This Rmd file with your answers.
  - A PDF version of this file (use the knit to PDF option above).
- You can also write math expression via LaTeX, using the dollar sign, for example β.
- You can navigate the sections of this file clicking (Top Level) in the bottom of RStudio's code window.

Intro1 - Loading packages and data

Loading some packages first. Do `install.packages()` for each package the first time you use a new package.

```
library(keras) # Package for data transformations and tables
library(caret) # For some useful tools like the confusion matrix function
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(MLeval) # for plotting ROC curves and more
library("RColorBrewer") # for pretty colors
colors = brewer.pal(12, "Paired")[c(1,2,7,8,3,4,5,6,9,10)];
options(repr.plot.width = 12, repr.plot.height = 12, repr.plot.res = 100) # plot size
set.seed(12332)          # set the seed for reproducability
```

Problem 1 - Fit a logistic regression to MNIST using keras The first part of this lab is concerned with predicting hand-written images using the famous MNIST data consisting of 60000 handwritten 28 x 28 pixel grayscale images with labels for training and another 10000 labeled images for testing. Let's load the data and set up training and test datasets:

```
mnist <- dataset_mnist()  # Load the MNIST data
x_train <- mnist$train$x  # Set up training and test images with labels
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

x_train <- array_reshape(x_train, c(nrow(x_train), 784)) # flatten images matrices to vectors
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
x_train <- x_train / 255 # rescale grayscale intensities from 0-255 to interval [0,1]
x_test <- x_test / 255
```

```
# One-hot versions of the labels (0-9)
y_train <- to_categorical(y_train, 10) # 60000-by-10 matrix, each row is one-hot
y_testOrig <- y_test # Keep the original 0-9 coded test labels.
y_test <- to_categorical(y_test, 10)
```

Use the `keras` package to fit a simple (linear) logistic regression to the training data. Use the cross entropy loss, the `rmsprop` optimizer, 30 epochs, batchsize 128, and monitor performance using accuracy on 20% of the data used for validation in the fitting. Is the model underfitting or overfitting?

Make predictions on the test data and compute the test accuracy. Use the `MLeval` package to compute the confusion matrix for the test data. Which digits is most frequently wrongly predicted to be a 2?

Problem 2 - Fit models with hidden layers to MNIST Add hidden layers now to the model in Problem 1. Fit and compute the accuracy on the test data for the following models 4 models:

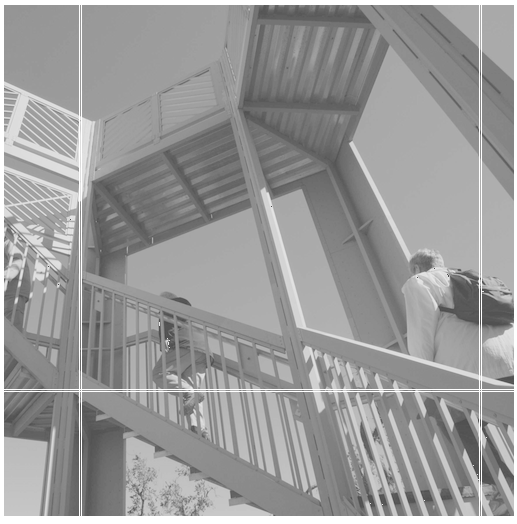
- Model with 1 hidden layer with 16 hidden units.
- Model with 1 hidden layer with 128 hidden units.
- Model with 3 hidden layer with 16 hidden units.
- Model with 3 hidden layer with 128 hidden units.

Let all layers hidden layers have `relu` activation functions and use the same settings as in the logistic regression in Problem 1 when fitting the models.

Which seems be the most important: deep models with many layers, or models with many hidden units in the layers?

Problem 3 - Filtering images As a pre-cursor to convolutional networks, here is a little exercise on filters and convolutions. Let's load a test image for this problem and plot it using the `image` function.

```
library(imagine)
library(pracma)
ascent = as.matrix(read.table(file =
  "https://github.com/mattiasvillani/MLcourse/raw/main/Data/ascent.txt", header = FALSE))
ascent = t(apply(ascent, 2, rev))
par(pty="s")
image(ascent, col = gray.colors(256), axes = F)
```



Apply the following filters to the image:

- Horizontal 3x3 Sobel edge detector

- Vertical 3x3 Sobel edge detector
- 15x15 Gaussian blur with a standard deviation of 3.

Code up the above filter matrices yourself, but you can use the `convolution2D` function from the `imagine` package for the convolution.

Problem 4 - Fit convolutional neural networks to CIFAR In this problem you will work with the CIFAR10 data, a dataset with 28x28 RGB color images from 10 labeled classes. The following code loads the data, scales it and plots one of the images.

```
# See ?dataset_cifar10 for more info
cifar10 <- dataset_cifar10()
classes = c('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
y_label_train = classes[cifar10$train$y+1]
y_label_test = classes[cifar10$test$y+1]

# Scale RGB values in test and train inputs
x_train <- cifar10$train$x/255   # x_train[1,,,1] is the red channel for first pic
x_test <- cifar10$test$x/255
y_train <- to_categorical(cifar10$train$y, num_classes = 10)
y_test <- to_categorical(cifar10$test$y, num_classes = 10)

# Let's only use 10000 of the 50000 images for training. Runs faster.
x_train <- x_train[1:10000,,,]
y_train <- y_train[1:10000,]
y_label_train <- y_label_train[1:10000]

# Plot an image and check if the label is correct
image_no = 20
rgbimage <- rgb(x_train[image_no,,,1], x_train[image_no,,,2], x_train[image_no,,,3])
dim(rgbimage) <- dim(x_train[image_no,,,1])
y_label_train[image_no]
```

```
## [1] "frog"
```

```
library(grid)
grid.raster(rgbimage, interpolate=FALSE)
```

Fit a convolutional neural network to the training data. Use at least two hidden convolutional layers, but otherwise you can experiment with different network structure and regularization (e.g. dropout) as you wish.

Compute the confusion matrix on the test data. Which classes are most easily mistaken by the classifier?

Problem 5 - Gaussian process regression for the bike share data This problem will fit a Gaussian process regression model to the bike share data from Lab 1a. We will be using only February of 2011 for training, and only the variable `hour` as predictor for `logrides`. Let's load the data and get started:

```
bikes = read.csv("https://github.com/mattiasvillani/MLcourse/raw/main/Data/BikeShareData/hour.csv")
bikes$dteday = as.Date(bikes$dteday) # convert date column to proper date format
bikes$logrides = log(bikes$cnt)      # we model the log(number of rides) as response.
bikes$hour = bikes$hr/23             # hour of the day. midnight is 0, 11 PM is 1.
bikesTrain = bikes[bikes$dteday >= as.Date("2011-02-01") &
                     bikes$dteday <= as.Date("2011-02-28"),] # Data from feb 2011
```

Consider now the Gaussian process regression:

$$y = f(x) + \varepsilon, \qquad \varepsilon \sim N(0, \sigma_n^2),$$

where y are the observed `logrides` and x is the observed `hour`. Fit a Gaussian process regression to the bike share data from February in 2011, using the squared exponential kernel. The noise standard deviation in the Gaussian process regression, σ_n, can be set equal to the estimated residual variance from a polynomial fit of degree 3. Use a zero (prior) mean for the function $f(x)$.

I want you to code everything from scratch. This involves coding:

- The squared exponential kernel function $k(x, x')$ for any two inputs x and x'.

- A function that evaluates the kernel function $k(x, x')$ over a dataset with n data points and returns the $n \times n$ covariance matrix $K(\mathbf{x}, \mathbf{x}')$.
- A function that computes the mean and standard deviation of the function f for a test set $\mathbf{x}_\star$ of input values.

The end result in your report should be a scatter plot of the data with the mean of f overlayed, as well as 95% probability bands for f. Note that this involves predicting on a test set with `hour` on a fine grid of values, e.g. `hourGrid = seq(0, 1, length = 1000)`.

The squared exponential kernel has two hyperparameters ℓ and σ_f. Experiment with different values and report your results for a set of values that seems to fit the data well and gives you a smoothing that you find pleasing to the eye. In practice one would estimate these hyperparameters, for example by maximizing the log marginal likelihood, but I do not require that here.