

Lecture 5: Artificial neural networks

Isabel Casas
icasas@deusto.es

Recapitulation of Machine Learning

- We assume there is a true relationship $y = g(x)$ to explain some phenomenon. Something perhaps very complex.
- For example, if x are images of dogs and cats, we humans know how to differentiate them, no matter their colour, shape or race. However, it is not an easy task for a machine learning model.
- So it has to learn from many pictures (data) to identify pictures of cats and dogs.
- Mathematically, we need to approximate this function g with a mathematical/statistical **model** $f(\tilde{x}; \theta)$, where \tilde{x} is most likely a subset of x and θ are parameters of the model f .

Recapitulation of Machine Learning

- We assume there is a true relationship $y = g(x)$ to explain some phenomenon. Something perhaps very complex.
- For example, if x are images of dogs and cats, we humans know how to differentiate them, no matter their colour, shape or race. However, it is not an easy task for a machine learning model.
- So it has to learn from many pictures (data) to identify pictures of cats and dogs.
- Mathematically, we need to approximate this function g with a mathematical/statistical **model** $f(\tilde{x}; \theta)$, where \tilde{x} is most likely a subset of x and θ are parameters of the model f .
- what is a model?

Recapitulation of Machine Learning

There are loads of different models f and each of them can have several parameters θ . In machine learning, we need to:

- 1 Choose the best model f for our problem (linear regression, random forest, hierarchical cluster, support vector machine, neural networks, etc)
 - 2 Not only f must be chosen, but also the best hyperparameters for our problem. The ones that approximate $f(\tilde{x}; \theta)$ to $y = g(x)$ best, i.e. minimises **cost function**.
- Examples of cost functions: MSE, MAE, error rate, etc.

Recapitulation of Machine Learning

Figure 1 displays the process involved in a machine learning problem.

Machine learning

learning patterns from data to predict other outputs or
make inference on the population

DATA	MODEL	COST FUNCTION	OPTIMISATION
<pre> age job marital education default balance 1 58 management married tertiary no 2143 2 44 technician single secondary no 29 3 33 entrepreneur married secondary no 2 4 47 blue-collar married unknown no 1586 5 33 unknown single unknown no 1 6 35 management married tertiary no 231 month duration campaign pdays previous outcome y 1 may 261 1 -1 0 unknown no 2 may 151 1 -1 0 unknown no 3 may 76 1 -1 0 unknown no 4 may 92 1 -1 0 unknown no 5 may 198 1 -1 0 unknown no 6 may 139 1 -1 0 unknown no </pre>	$f(\tilde{x}, \theta) \rightarrow y = g(x)$ <p>g - true function x - true inputs/features f - model \tilde{x} - sample inputs θ - parameters</p>	$C(f(\tilde{x}_{train}, \theta), y)$ <p>Examples C: MSE, MAE, likelihood function, etc</p>	$\frac{\partial C(f(\tilde{x}_{train}, \theta), y)}{\partial \theta}$ <p>Find the parameters that minimize the cost function</p>

Figure 1: Summary of machine learning process.

Tuning the model - finding hyperparameters

It is easy to follow the machine learning process when using package *caret*. It gives us the possibility of doing many things with only one package:

- 1 Preprocess the predictors (imputation with knn, scaling, centering, reduction of dimensionality with PCA, etc)
- 2 Split the data using different strategies like *cv*, *bootstrapping*, etc
- 3 Automatic tuning of hyperparameters for optimal model performance
- 4 Choose the optimal model based on a given evaluation metric

History of machine learning: Can machines think?

You can read the details in [here](#)

- 1949: Brain cell interaction model (Donald Hebb) - neuroscience model of the brain
- 1950: Can machines think? (Alan Turing) - arguably the first [paper](#) that questions this concept. He also was involved in the Turing machine.
- 1956: Program to play checkers in an IBM 701 (Arthur Samuel) - among the first successful learning programs.
- 1959: Coinage phrase “machine learning” (Arthur Samuel) - [paper](#)

History of machine learning: Can machines think?

- 1957-1962: Perceptron (Frank Rosenblatt) - first neural network (NN) algorithm. Marvin Minsky and Symour Papert outlined the limitations at the time of NNs in 1969.
- 1967: Nearest neighbours algorithm - KNN ([Cover and Hart, 1967](#))
- 1986: The idea of backpropagation errors was posed in the 60s but it was not until 1986 that it is used in neural networks - it is a standard method to train artificial neural network. It is simple and easy to program.
- 1980s: Separation of machine learning and artificial intelligence. The former focuses in using models to provide services. The latter focuses in training machines to do different tasks. In this period ML struggles to survive until the arrival of the Internet and massive data

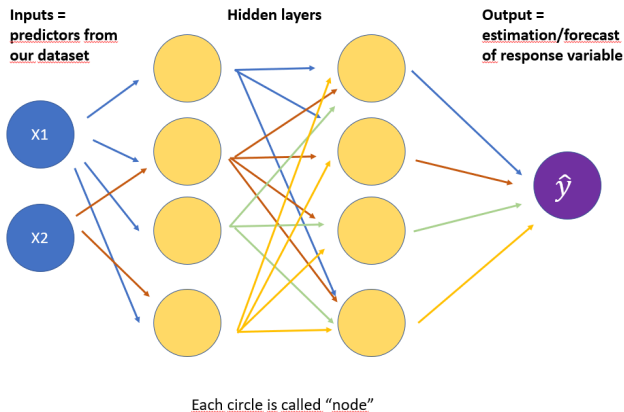
History of machine learning: Can machines think?

- 1990: Boosting algorithm (Robert Shapire) - reduce the bias during supervised learning by using a set of weak learners to create a single strong learner. Examples: AdaBoost, BrownBoost, LPBoost, MadaBoost, etc.
- 1995: Support vector machine (Corinna Cortes and Vladimir Vapnik).
- 1997: Long short-term memory (LSTM, Sepp Hochreiter) - a type of deep neural network commonly used for speech recognition training.
- 2006: the **Face Recognition Grand Challenge**.
- Now: ML is being used for self-driving vehicles, exploring the universe, new algorithms for robots, analytical tools, chatbots, and more.

Artificial Neural Networks

- We know there are problems that cannot be solved linearly and ANN is a powerful tool to deal with this kind of problems.
- They satisfy the **universal approximation theorem**, meaning that theoretical they can approximate any bounded function but adding the necessary hidden layers.
- The simplest type of ANN are the Feed-forward neural networks (FFNN) [Zell1994]. They consist of an input layer, a number of hidden layers and an output layer. All the calculations are done sequentially. In contrast to Recurrent neural networks where they can be cycles in the process.

Feed forward neural network (FFNN)



Watch Video: [Feed-forward Neural Networks](#)

Types of FFNN

- Single-Layer Perceptron: input + output neurons. It is used for binary classification tasks and can only learn linearly separable patterns.
- Multilayer Perceptron (MLP): input + 1 or more hidden layers + output. MLPs are used for classification, regression, and pattern recognition.
- Radial Basis Function (RBF) Networks: Uses radial basis functions as activation functions. RBF networks are used for pattern classification, clustering, and function approximation tasks.
- Cascade Correlation Networks: These are a type of FFNN that are trained in a hierarchical manner, with new neurons added to the network one at a time. Cascade correlation networks are often used for function approximation tasks.
- Self-Organizing Maps (SOMs): These are a type of FFNN that are used for clustering and dimensionality reduction tasks. SOMs use unsupervised learning to learn a low-dimensional representation of high-dimensional data.

Multilayer perceptron (MLP)

A Multilayer perceptron with one hidden layer looks like a linear model transformed by an activation function:

$$z_1 = \sigma_1 \left(\sum_j (w_{1,j} x_j + b_{1,j}) \right) \text{ (hidden layer)} \quad \hat{y} = \sigma_2 \left(\sum_j w_2 z_1 + b_2 \right) \text{ (output)}$$

where x_j are the input units, $w_{1,j}$ are the weights and $b_{1,j}$ the bias of the hidden layer.

Isabel: plot that particular MLP

Multilayer perceptron

Functions $\sigma_1()$, $\sigma_2()$ are nonlinear activation functions:

- Sigmoid: $\sigma(z) = \frac{1}{1 + e^{-z}}$ (logistic function)
- ReLU: $\sigma(z) = \max(0, z)$
- Tanh (hyperbolic tangent): $\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- Softmax: $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$, where z_i is the input to the function for class i , K is the total number of classes, and the sum is taken over all classes j . The output of the softmax function is a probability distribution over the classes.

Activity 1

How are the mathematics of MLP with two layers? Plot and write the formula.

Hyperparameters of MLP

- The number of units in a layer is called **width**.
- The number of layers is called **depth** or **size**.
- Activation function.
- Learning rate or **decay**.
- The **batch size** determines the number of samples used to compute the gradient during each training iteration.
- Regularization techniques such as **L1** and **L2** regularization can be used to prevent overfitting of the model to the training data.
- Dropout is a regularization technique that randomly drops out some of the hidden units during training. The **dropout rate** is a hyperparameter that controls the degree of dropout.

Activity 2: MLP in *caret*

The *caret* package supports several types of ANN.

Find the name of the methods in *caret* that fit MLPs and check their hyperparameters.

Example MLP: MPG

As upload the cleaned dataset. I am going to use a bootstrap strategy in the data splitting because my dataset is very small and I am going to fit a randomForest and MLP models, see which work better.

```
library(caret)
mpg.data <- read.table("../Lecture02/data/mpg_new.csv",
                        header = TRUE, sep = ";")

set.seed(1234)
# train/test split 80-20 percent
training_indexes <- createDataPartition(mpg.data$mpg,
                                         p = 0.8, list = FALSE)
training <- mpg.data[training_indexes, 1:7]
testing <- mpg.data[-training_indexes, 1:7]
```


Example MLP: MPG

Now, we are fitting a MLP. We are using data *training* to train the model and also, we are using bootstrapping to have a few samples of training to get the best model possible. The model will tune hyperparameter *size*.

The chunk below:

- 1 Set the model to use 100 bootstrap samples for training
- 2 It trains a MLP using *method* = "mlp" which has one hyperparameter (*size*) which is tuned using a grid of values

```
library(RSNNS)
control <- trainControl(method = "boot", number = 100,
                        returnData = FALSE,
                        savePredictions = "final")
mpg.mlp <- caret::train(mpg ~ ., data = training, method = "mlp",
                       trControl = control,
                       tuneGrid = data.frame(size = 1:7))
```


Example MLP: MPG

We see that the best performance in the sense of lower RMSE occurs for a MLP with 4 hidden layers.

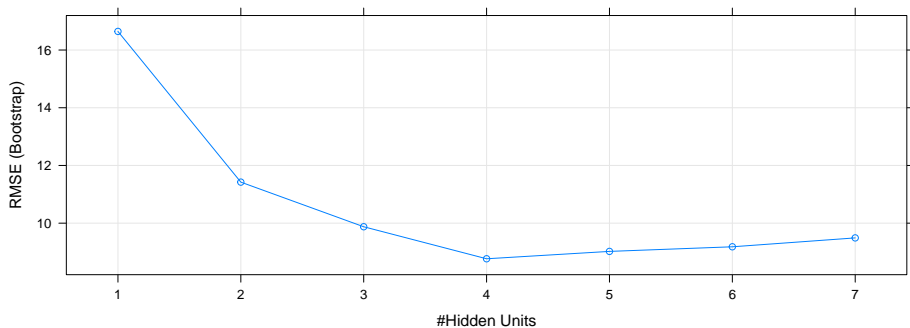
mpg.mlp

```
## Multi-Layer Perceptron
##
## No pre-processing
## Resampling: Bootstrapped (100 reps)
## Summary of sample sizes: 321, 321, 321, 321, 321, ...
## Resampling results across tuning parameters:
##
##   size  RMSE      Rsquared   MAE
##   1     16.644298  0.03486579  15.385578
##   2     11.422404  0.02313107  10.108947
##   3      9.876160  0.01934356   8.538618
##   4      8.766534  0.01158543   7.349362
##   5      9.021877  0.01410858   7.573056
##   6      9.181651  0.01269743   7.640271
##   7      9.489939  0.01381133   7.848650
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was size = 4.
```


Example MLP: MPG

This shows error versus different value of hyperparameter

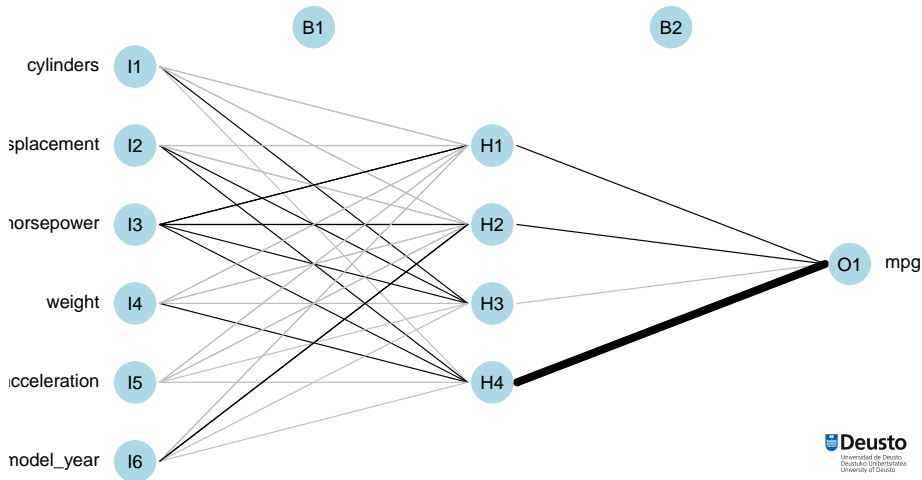
```
plot(mpg.mlp)
```



Example MLP: MPG

Plot structure of NN using package **NeuralNetTools**.

```
library(NeuralNetTools)
plotnet(mpg.mlp)
```



Example MLP: MPG

NN backbone:

```
extractNetInfo(mpg.mlp$finalModel)$infoHeader
```

##	name	value
## 1	no. of units	11
## 2	no. of connections	28
## 3	no. of unit types	0
## 4	no. of site types	0
## 5	learning function	Std_Backpropagation
## 6	update function	Topological_Order

Example MLP: MPG

Weights in each hidden layer

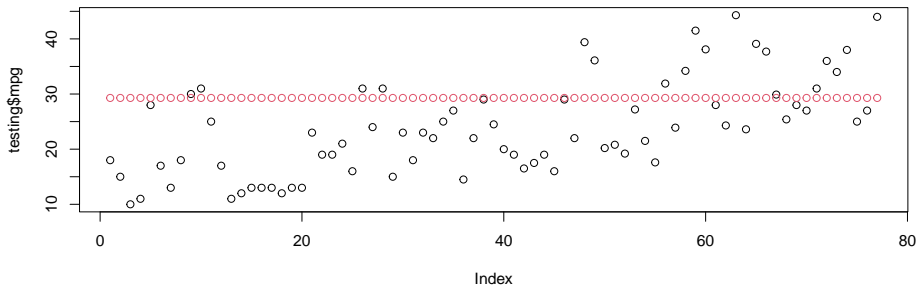
```
extractNetInfo(mpg.mlp$finalModel)$fullWeightMatrix[, 7:11]
```

	Hidden_2_1	Hidden_2_2	Hidden_2_3	Hidden_2_4	Output_1
## Input_cylinders	-0.11404	-0.28799	0.21397	-0.20841	0.00000
## Input_displacement	-0.23789	-0.11302	0.17714	0.19299	0.00000
## Input_horsepower	0.08398	0.14122	0.01204	0.01882	0.00000
## Input_weight	-0.13652	-0.11344	-0.06680	0.15117	0.00000
## Input_acceleration	-0.14259	-0.19610	-0.12172	-0.10899	0.00000
## Input_model_year	-0.26276	0.22354	-0.29905	-0.09789	0.00000
## Hidden_2_1	0.00000	0.00000	0.00000	0.00000	0.21749
## Hidden_2_2	0.00000	0.00000	0.00000	0.00000	0.26886
## Hidden_2_3	0.00000	0.00000	0.00000	0.00000	-0.29672
## Hidden_2_4	0.00000	0.00000	0.00000	0.00000	29.29231
## Output_1	0.00000	0.00000	0.00000	0.00000	0.00000

Example MLP: MPG

Prediction

```
mpg.predict <- predict(mpg.mlp, newdata = testing)
plot(testing$mpg)
points(mpg.predict, col=2)
```



Activity 3

Compare the prediction of *mpg* using MLP and random forest. Which model does better, RF or MLP?