Supervised Learning - Neural Networks (NN)

Mark Asch - IMU/VLP/CSU

2023

Program

1. Data Analysis
   (a) Introduction: the 4 identifiers of “big data” and “data science”
   (b) Supervised learning methods: regression—advanced, k-NN, linear classification methods, SVM, NN, decision trees.
   (c) Unsupervised learning methods: k-means, principal component analysis, clustering.

Introduction

- A neural network (NN) is a supervised learning model for classification or regression,
  - ⇒ We use the data/observations/measurements to train the NN.
  - ⇒ The NN learns a functional relation between inputs, X, and outputs, Y.
  - ⇒ The NN in fact learns an approximation to the relation by iterative fitting of its parameters.

- The simplest model of an NN is the multilayer perceptron (MLP that has the impressive property of being a universal approximator---see below.
  - ⇒ It can be proved that MLPs, as universal approximators, can model any suitable smooth function, given enough hidden units, to any desired level of precision.
  - ⇒ These hidden units are made up of (linear combinations of) nonlinear activation functions whose parameters are "hidden" from the model, in that they are intermediary, unobserved variables in the model.

⇒ The network is a directed graph with an architecture made up of
  - → layers,
  - → nodes ("neurons") and
  - → activation functions.

- NNs can become extremely complex and they tend to overfit.

- They must thus be handled with extreme care, but when they work, the results are amazing!

Structure

- The NN seeks to learn a mapping, f, from the input layer, $\mathbf{x}$, into the output layer $\mathbf{y} = f(\mathbf{x})$.

- A layer is defined as a set of neurons having no connections between them.

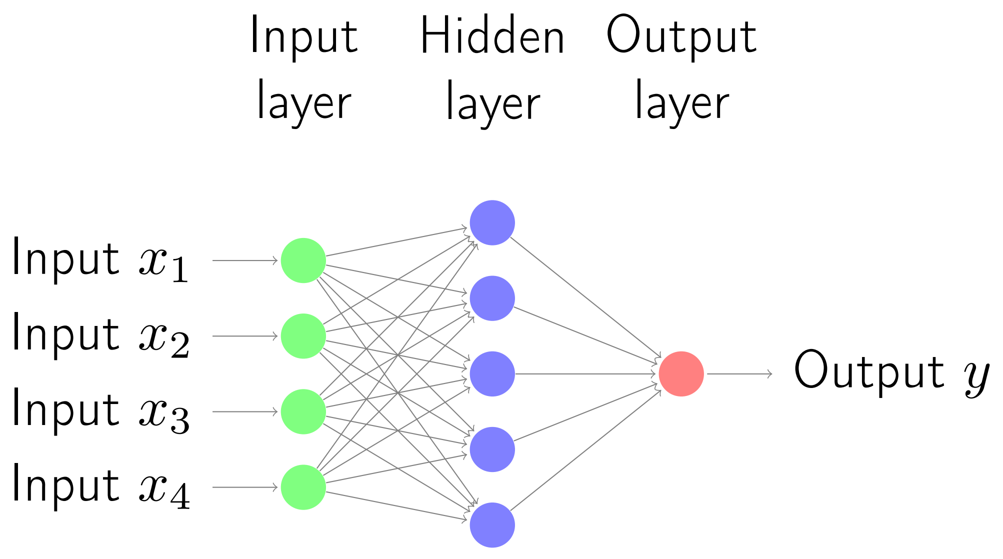
  $\Rightarrow$ one layer reads the input signals, with a single neuron per input x_i, $i = 1, \ldots, 4$

  $\Rightarrow$ the output layer provides the response of the system, y,

  $\Rightarrow$ one or more hidden layers, between the two, perform the approximation.

- A neuron is a model of a biological neuron—it receives input signals, computes a weighted sum of the inputs, applies an activation function and finally produces an output.

Neurons and Layers



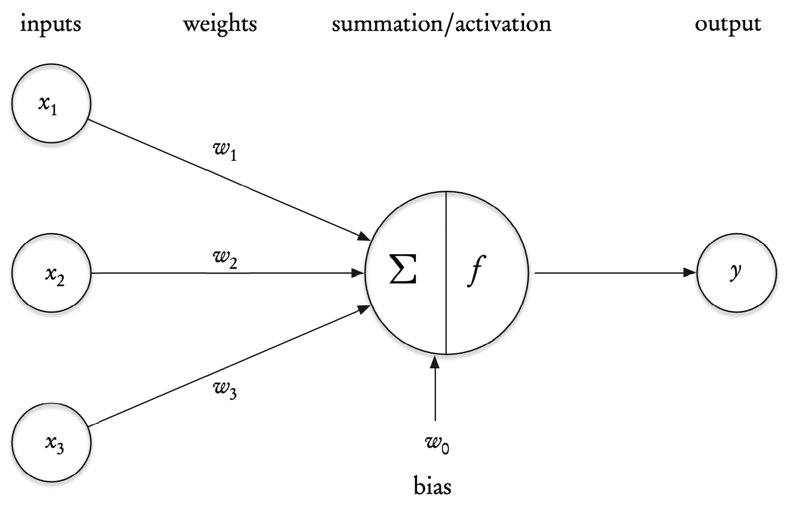


Figure 1: A single neuron with output $y = f\left(w_0 + \sum_{i=1}^{3} w_i x_i\right)$

Networks with several layers

- For simplicity, suppose that we have a linear mapping between the layers, then $\mathbf{x}^{(1)} = A_1\mathbf{x}$, and $y = A_2\mathbf{x}^{(1)}$.
- The overall structure is then a composition of linear operators $y = A_2 A_1 \mathbf{x}$.
- This can be generalized to M nonlinear mappings, and to a vector output,

$$\mathbf{y} = f_M\left(A_M, \ldots, f_2(A_2, f_1(A_1\mathbf{x}))\cdots\right).$$

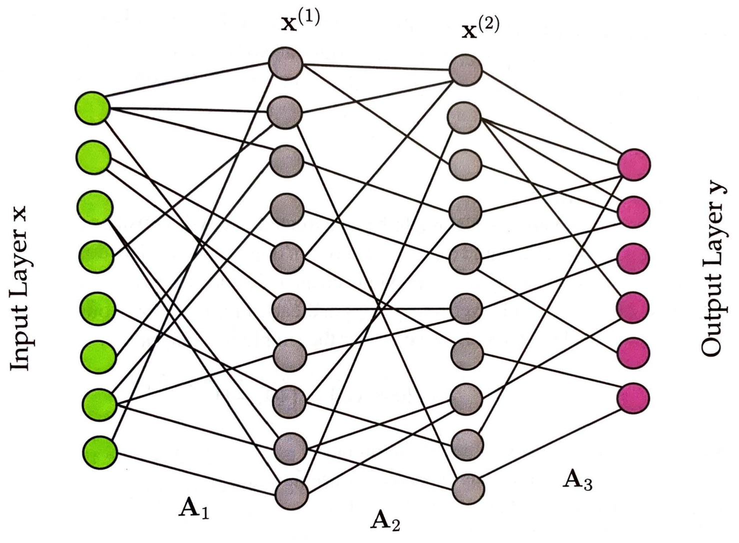


Figure 2: Neural network with 2 hidden layers.

Formulation

- Then, as in all the rest of ML, the objective is to find the unknown model parameters by optimizing. In this case, the system is highly under-determined (there are far fewer equations than unknowns) and some type of regularization will be necessary.

- The problem then is to compute

$$\underset{A_j}{\operatorname{argmin}} \left[f_M \left(A_M, \ldots, f_2(A_2, f_1(A_1\mathbf{x})) \cdots \right) + \lambda g(A_j) \right],$$

  which is an underdetermined system regularized by $\lambda g(A_j)$, where λ is a regularization coefficient and g is a smoothing function.

- This is usually performed by a stochastic gradient plus backpropagation algorithm (see Advanced Course).

- This should, as always, be followed systematically by cross validation.

Activation Functions

- In the sought for mapping from input to output space, we will write

$$\mathbf{y} = f(\mathbf{A}, \mathbf{x})$$

  and f will be called the activation function, borrowing from the neuroscience analogy.

- Nonlinear activation functions are used, since linear mappings (even composed) can only provide a limited range of functional responses.

- The principal ones are,

$$\begin{aligned}
f(x) &= x, \quad \text{linear}, \\
f(x) &= \mathbf{1}_{0,+\infty}(x), \quad \text{threshold}, \\
f(x) &= \begin{cases} 0, & x \leq 0 \\ x, & x > 0, \end{cases} \quad \text{ReLU}, \\
f(x) &= \tanh(x), \quad \text{hyperbolic tangent}, \\
f(x) &= 1/(1+e^{-x}), \quad \text{soft step}.
\end{aligned}$$

- The function f performs a transformation of an affine combination of input signals

$$\mathbf{y} = f(\mathbf{x}) = f(\alpha_0 + \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{x}),$$

where α_0 is the bias of the neuron, $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_p]^{\mathrm{T}}$ is a vector of weights associated to each neuron.

$\Rightarrow$ These are the values that are estimated in the training phase and they constitute the "memory", or distributed knowledge of the network.

$\Rightarrow$ The two most commonly used activation functions are the Rectified Linear Unit (ReLU), the sigmoid (soft-step) and the hyperbolic tangent (tanh).

Training

- The basis for the learning phase is made up of
  - ⇒ the n observations $\{x_i^1, \ldots, x_i^p; y_i\}_{i=1}^n$,
  - ⇒ the p explanatory variables (covariates) $X^1, \ldots, X^p$ and
  - ⇒ the (response) variable to be predicted Y.
- Let us consider, as a simple example, the case of regression with a network made up of one neuron with linear output and a hidden layer of q neurons whose parameters will be computed by minimization of the least squares criterion (loss function)

$$Q(\alpha, \beta) = \sum_{i=1}^{n} [y_i - f(x; \alpha, \beta]^2 ,$$

  where β are the regression coefficients
- This minimization is usually performed by variants of stochastic gradient with backpropagation—see the Advanced Course for details.

Choice of Parameters

To perform this training, the user must follow these steps:

1. Input and output variables must, as in most statistical treatments, be transformed, centered and scaled to avoid too large disparities and over sensitivity to small errors in the estimation process.

2. Network architecture must be designed:

   (a) Choose the number of hidden layers that will determine the aptitude of the network to deal with nonlinearity.

   (b) Choose the number of neurons per hidden layer.

   (c) These two choices will directly condition the number of parameters (weights) to estimate and thus the model complexity. They contribute to the search for a reasonable compromise between bias and variance that determines the balance between training quality and forecast quality.

3. Three additional parameters influence the above compromise:

   (a) the maximum number of iterations,
   (b) the maximum error tolerance, and
   (c) an eventual regularization term for ridge decay (to eliminate terms with"weak" influence on the parameters—see previous lecture on Regression.)

4. Choose the training/learning rate or the stochastic gradient as well as an evolution strategy for it—see Advanced Course.

5. Choose the size of the sets, or batches, of observations that are to be considered at each iteration.

- In practice, all of these parameters cannot be tuned simultaneously by the user. One has to, in priority, make choices regarding the control of overtraining, or overfitting, by

  $\Rightarrow$ limiting the number of neurons, or the duration of the training,

- $\Rightarrow$ or increasing the penalization of the magnitude of these parameters.

- To do this, one needs to fix a method for estimating the error, by using either
  - $\Rightarrow$ a validation sample,
  - $\Rightarrow$ or a form of cross validation (including bootstrapping).

- A simple, but efficient strategy is to introduce a relatively large number of neurons, then optimize the unique regularization (decay) parameter by cross-validation. These important issues of tuning and cross-validation will be discussed in a dedicated lecture.

Universal Approximation Property

- The reason, at least theoretically, why neural networks can be so effective are their universal approximation properties.

- These theoretical results [Pinkus,1999] show that any functional relationship, f, can be approximated, to within a desired tolerance, by a neural network of sufficient depth (number of hidden layers) and width (number of nodes in a layer).

- A very nice visual proof can be found at `http://neuralnetworksanddeeplearning.com/chap4.html`.

Pros and Cons

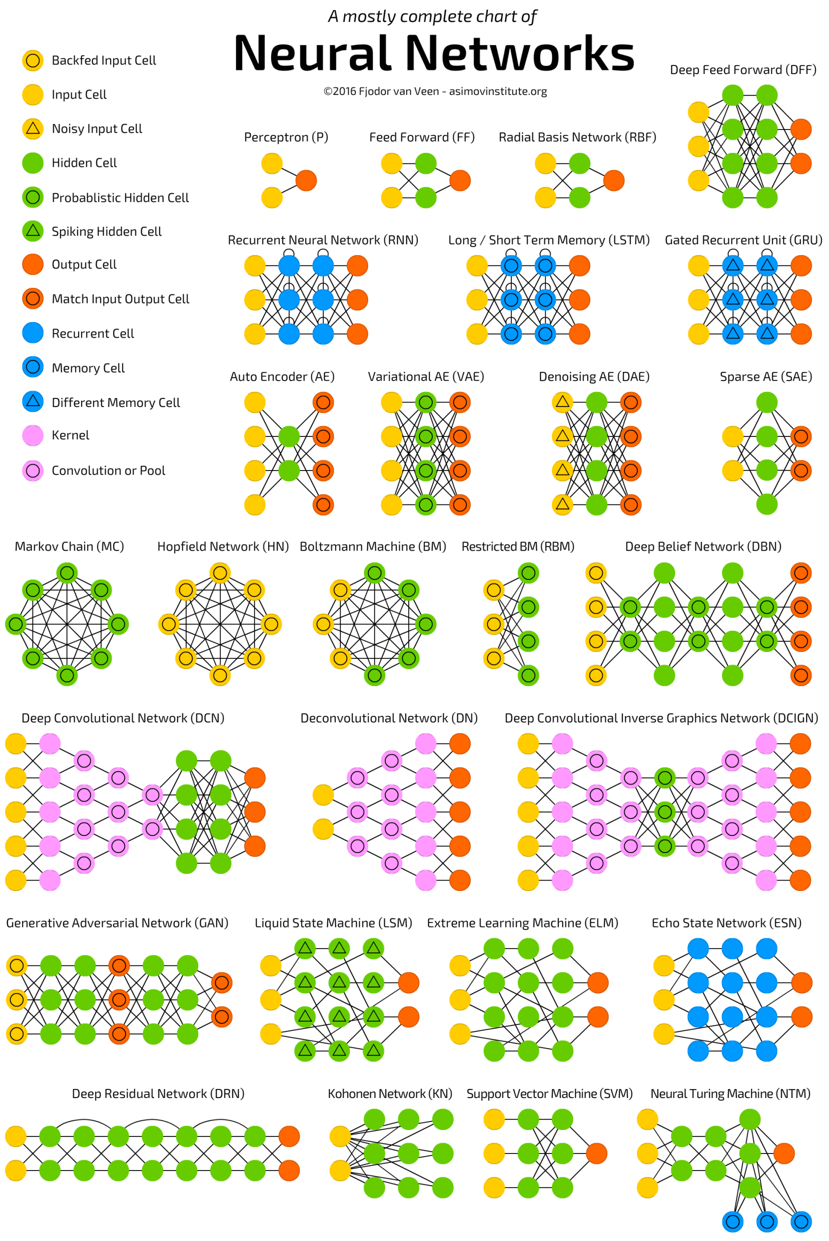
- ✔ The domains of application of MLPs are numerous. They cover discriminants, forecasting of time series, shape recognition, and many others. These are generally well explained in the documentation of the specialized software packages.

- ✘ The main criticisms expressed, concern the difficulties related to the training (computing time, sample size, local minima) and the "black box" status of neural networks, in general. In fact, contrary to discriminant or tree models, it is impossible to know the effective influence of an input variable on the system as soon as a hidden layer intervenes. Nonetheless, search techniques for system sensitivity to each input can enable a focusing of the model and eventually a simplification of the system by suppressing certain inputs. This feature selection is often necessary.

- ✔ On the other hand, MLPs have undeniable qualities when there is no linearity and when the absence of explanatory variables makes traditional statistical models ineffective or unusable. The flexibility of NNs, thanks to the possibility of inserting specific layers (especially in deep learning[Goodfellow2016], coupled with a training process that integrates weighting and choice of both variables and their interactions, can make these networks extremely effective.

- ✘ The downside of this flexibility is the risk, and the general tendency, of neural network to overfit the data. As a result the statistical models produced are often very brittle and suffer from a lack of explainability, or interpretability.

Other NNs


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

- Perceptrons (mulit-layer), fully-connected, feed-forward - the basis...
- Convolutional Networks (CNN) - LeCun, image recognition
- Recurrent Networks (RNN) - generalization of MLP
- LSTM (long short term memory) - for time series
- Hopfield Networks - dynamical systems
- Boltzmann Machines - stochastic networks
- Autoencodeurs - unsupervised training, similar to PCA
- GAN (adversarial networks) - combination of 2 networks (generative, discriminative)
- Transformers - basis of LLMs (ChatGPT)

MLP 1: structure

- MLPs can be applied to modeling any functional relationship thanks to the "universal approximator" property of these neural networks.
- The MLP is structured as a directed graph[1] made up of nodes (or "neurons") and directed arcs (the "synapses").
- The neurons, as shown above, are organized into layers that are completely connected by the synapses between two layers, but not within the layer.
- As seen above, there are three types of layers:

  $\Rightarrow$ Input layer made up of all the covariates, each in a separate neuron.
  $\Rightarrow$ Output layer made up of all the response variables.
  $\Rightarrow$ Hidden layers made up of an arbitrary number of neurons.

[1]We also encounter these in Bayesian networks and causality analysis.

- Note that the input and hidden layers contain a constant neuron that is not influenced by (connected to) the covariates. This neuron plays the role of the bias, or constant term in the regression.

MLP 2: single-layer example

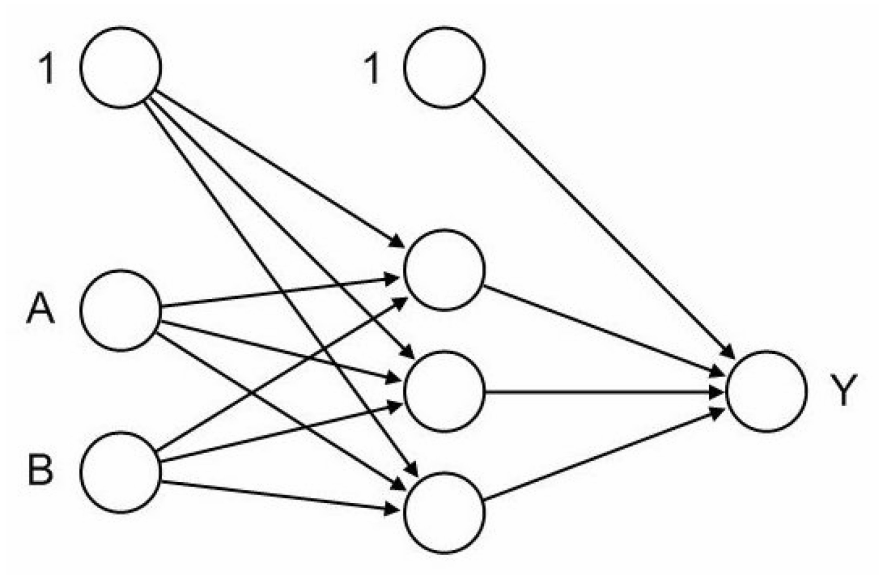


Figure 3: Single-layer MLP between inputs A and B, and output Y. The digit 1 denotes the constant term, or bias.

- MLP with one hidden layer containing three neurons.
- This neural network models the relationship between two explanatory variables, A and B, and the scalar response variable Y.
- A weight, w, is attributed to each synapse, indicating

the effect of the corresponding neuron, as if all the data were passing through the NN as signals.

- The signals are then analyzed by

  ⇒ the integration function that combines all entering signals,

  ⇒ then by the activation function that transforms the output of each neuron.

MLP 3: model formulation

- The simplest MLP consists of an input layer with n covariates, and an output layer with a single output neuron,

$$o(x) = f\left(w_0 + \sum_{i=1}^{n} w_i x_i\right) = f\left(w_0 + \mathbf{w}^{\mathrm{T}}\mathbf{x}\right),$$

  where w_0 is the intercept, $\mathbf{w} = (w_1, \ldots, w_n)$ is the vector of synapse weights, and $\mathbf{x} = (x_1, \ldots, x_n)$ is the vector of all the explanatory variables.

- To increase the flexibility of the model, hidden layers can then be added. In fact it can be proved that a single layer is sufficient for modeling any piecewise continuous function.

- For a model with one hidden layer of J hidden neurons, we have

$$\begin{aligned} o(x) &= f\left(w_0 + \sum_{j=1}^{J} w_j \cdot f\left(w_{0j} + \sum_{i=1}^{n} w_{ij} x_i\right)\right) \\ &= f\left(w_0 + \sum_{j=1}^{J} w_j \cdot f\left(w_{0j} + \mathbf{w}_j^T \mathbf{x}\right)\right), \end{aligned}$$

where w_0 is the intercept of the output neuron, w_{0j} the intercept of the j-th hidden neuron.

- Formally, all the hidden neurons and the output neuron calculate

$$f\left(g(z_0, z_1, \ldots, z_k)\right) = f\left(g(\mathbf{z})\right)$$

from the outputs of all the preceding neurons, $z_0, z_1, \ldots, z_k$ where $g : \mathbb{R}^{k+1} \to \mathbb{R}$ is the integration function and $f : \mathbb{R} \to \mathbb{R}$ is the activation function.

- Note that the neuron $z_0 = 1$ is the constant neuron attached to the intercept.

- The integration function is often defined by

$$g(\mathbf{z}) = w_0 + \sum_{i=1}^{k} w_i z_i = w_0 + \mathbf{w}^T \mathbf{z}.$$

- The activation function, f, is a bounded, non-decreasing, nonlinear and differentiable function that should be chosen in relation with the response variable.

$\Rightarrow$ For example, the logistic function is recommended for variables with a binary response.

The `neuralnet` package of R

- Implements training of «Multi-Layer Perceptrons» (MLP) to learn the relation between explanatory variables («covariates») and responses.
- Uses a backpropagation method.
- Can treat an arbitrary number of input and output variables.
- Can include an arbitrary number of hidden layers.

MLP 4: supervised learning

- The `neuralnet` function of R uses supervised learning algorithms where
  - ⇒ the given output is compared with the computed value,
  - ⇒ then the weights are adapted with respect to the comparison results.
  - ⇒ The weights are initialized by Gaussian random values.
- Then the iterative training process is:
  - ⇒ The NN calculates an output $o(\mathbf{x})$ for the given inputs, $\mathbf{x}$, and the actual weights;
  - ⇒ if the learning process is not finished, the predicted output, o, will differ from the observed output, y.
  - ⇒ The error is measured by a function, E, as the sum of squared errors (SSE)

$$E = \frac{1}{2} \sum_{l=1}^{L} \sum_{h=1}^{H} (o_{lh} - y_{lh})^2$$

or the cross-entropy

$$E = -\sum_{l=1}^{L}\sum_{h=1}^{H}\left(y_{lh}\log(o_{lh}) + (1-y_{lh})\log(1-o_{lh})\right),$$

where l is the index of the observations and h that of the observation nodes.

- The weights are adjusted according to the rules of the learning algorithm—usually stochastic gradient plus backpropagation (SGD + backprop).

MLP 5 : use of `neuralnet` - arguments

- There are two essential arguments:
  - ⇒ (1) the formula (of type `lm`): `response ~ covariate`,
  - ⇒ (2) a dataset that contains the covariates and the responses.
- All the other arguments have default values.
- Training:
  - ⇒ We can define the numbers of hidden layers and neurons per layer.
  - ⇒ The default value is a single hidden layer.
- The most important arguments are:
  - ⇒ `formula`: a symbolic description of the model to be fitted (no default value).
  - ⇒ `data`: a dataframe that contains the data specified in `formula`.

- ⇒ `hidden`: a vector defining the number of hidden layers and the hidden neurons in each layer (by default 1).
- ⇒ `threshold`: a numerical value defining the stopping criterion for the partial derivatives of E with respect to w_i (by default 0.01).
- ⇒ `rep`: number of repetitions for the learning process (by default 1).
- ⇒ `err.fct`: differentiable error function, `sse` (sum squared error) or `ce` (cross entropy error)—by default `sse`.
- ⇒ `act.fct`: differentiable activation function, `logistic` or `tanh`—by default `logistic`.
- ⇒ `linear.output`: logical value that determines if `act.fct` should not be applied to the output neurons—by default `TRUE`.
- ⇒ `likelihood`: logical value equal to `TRUE` if the error function is the negative log-likelihood function that computes the information criteria of Akaike (AIC) and Bayes (BIC)—by default `FALSE`.

- Note that if the response variable is **binary**, one must choose:

  ⇒ activation function: logistic,

$\Rightarrow$ error function: cross entropy,

$\Rightarrow$ `linear.output` must be `FALSE` so that the output is mapped by the activation function into the interval $[0, 1]$.

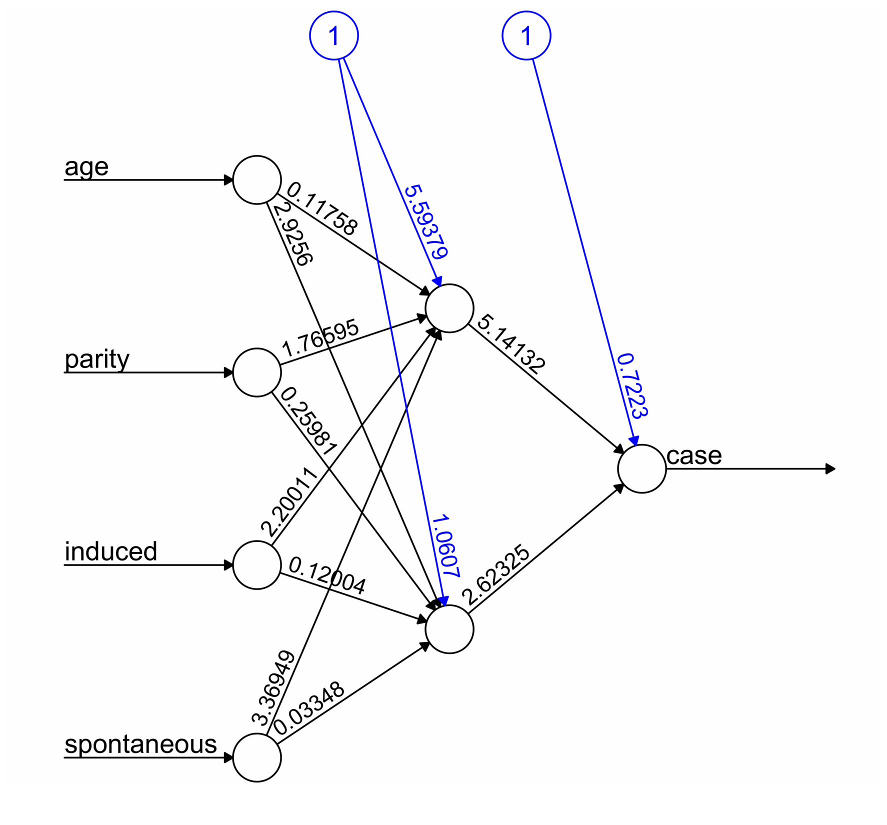
MLP 6: use of `neuralnet` - outputs

- Basic information on the learning process and the trained network are stored in the direct output of the call to `neuralnet`.
- `net.result` is a list that contains the output of the NN for each replication.
- `weights` is a list that contains the fitted weights.
- `result.matrix` is a matrix that contains the error, threshold, number of steps, AIC and BIC.

MLP 6: use of neuralnet - visualization

- The results of the training can be visualized by

$\Rightarrow$ `> plot(nn)`



Error: 125.212685 Steps: 5254

- The plot shows:
  - ⇒ the topological structure of the network,
  - ⇒ the synaptic weights after training,
  - ⇒ all the intercepts,
  - ⇒ the global error and the number of steps required for convergence.

Examples

1. Toy 2D classification : `https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html`
2. Synthetic data (simple example) : `nnet-squares.html`
3. STock exchange data: `nnet-dividendes.html`
4. Fuel consumption data: `nnet-essence.html`
5. Wine data: `nnet-MLP-wine.html.`

References

1. I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning.* MIT Press. 2016.

   http://www.deeplearningbook.org

2. Rachel Schutt and Cathy O'Neil. *Doing Data Science.* O'Reilly. 2014.