Neural Networks and Backpropagation

Alipio Jorge (DCC-FCUP)

November 2020

What is a Neural Network?

- When our aim is to learn a classification or a regression model
  - we want a **function** $\hat{f}$ that approximates the unknown function
- **How** do we define this function?
  - A linear function
  - By analogy of similar cases
  - By maximizing estimated probabilities
  - Using a decision tree (or a **regression tree**)
  - etc.

What is a Neural Network?

- A Neural Network is **another way** of defining functions
  - can be graphically described
  - but it always corresponds to a mathematical function
- Neural Networks are **flexible** and **powerful**
  - but not for all types of data

What is a Neural Network?

- **Many types** of networks and NN components
    - the Perceptron
    - the Multi-layer Perceptron
    - the Feed-Forward Network
    - Convolutional Neural Networks
    - Recurrent Neural Networks
    - LSTM, BiLSTM, GRU, GAN, ...
    - and multiple combinations of the above

The Perceptron

- A **perceptron** is the simplest and most fundamental **NN unit**
  - inspired in biological neurons
- It can define **simple functions**

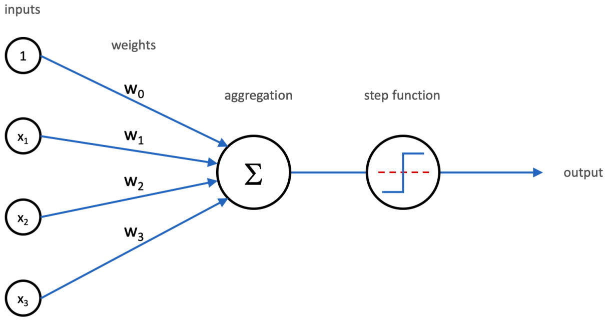


Figure 1: Binary Perceptron

The Perceptron

- Mathematically

$$\hat{y} = step(w_0 + \sum_{i=1}^{k} w_i . x_i)$$

- the **step** function gives a binary output depending on **threshold**

$$step(x) = \begin{cases} 0 & x < \theta \\ 1 & x \geq \theta \end{cases}$$

The Perceptron

- The step is an **activation function**
  - decides if the **neuron** fires or not
- The **sigmoid** is another activation function
  - very popular
  - good mathematical properties (to see later)

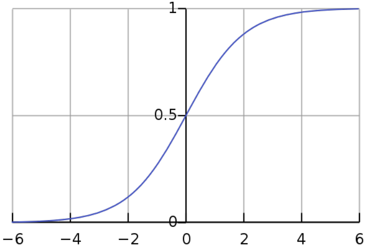


Figure 2: from wikipedia

Activation function sigmoid

- The perceptron becomes a **numerical** function
  - that can be used for classification

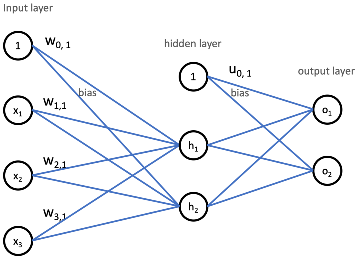
$$\hat{y} = sigmoid(w_0 + \sum_{i=1}^{k} w_i.x_i)$$

- sigmoid is defined as

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

The Multi-layer perceptron

- Side by side preceptrons
  - **multivalued** functions (non-binary classification)
- Adding a **hidden layer**
  - **fully connected**: each node linked to all nodes in nearby layers
  - more **expressive** functions
  - **abstraction** layers
- **bias** weights (intercepts)



The Multi-layer perceptron

- Mathematically
  - although commonly represented as a graph, a NN is a **mathematically** defined function
  - A two layer example: calculate hidden layer

$$h_j = activ(w_0 + \sum_{i=1}^{m} w_{ij}^{x}.x_i)$$

- hidden layer

$$h_j = activ(w_{0j}^{h} + \sum_{i=1}^{m} w_{ij}^{h}.x_i) \quad j \in \{1, \ldots, m_{hidden}\}$$

The Multi-layer perceptron

- output layer

$$o_j = activ(w^o_{0j} + \sum_{i=1}^{m_h} w^o_{ij}.h_i) \quad j \in \{1, \ldots, m_{out}\}$$

The Multi-layer perceptron

- *classical* ANN (Artificial Neural Networks) are MLP
  - Example with `irisdata` set
    - 4 predictors
    - 3 classees
  - We define previously the **topology** of the network
    - how many layers, how many nodes
  - The learning task is to find the best values for the **weights**
    - we say **learning** the weights
    - parameter **fitting**
    - **training the network**

The Multi-layer perceptron

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
  train_test_split(X, y, stratify=y,random_state=1)

clf = MLPClassifier(random_state=1,
                    hidden_layer_sizes=(8,),
                    max_iter=500,
                    activation='logistic'
                   ).fit(X_train, y_train)
```

The Multi-layer perceptron

- The MLP in this example
- **Nodes** or **units**
  - 4 input
  - 8 hidden
  - 3 output
- **Weights** (including **bias** weights)
  - $5 \cdot 8 + 9 \cdot 3$
- **Activation function**
  - sigmoid (logistic)
- **Class** is given by the highest output (of the three)

Feedforward networks

- MLP are **feedforward networks**
  - they can have many hidden layers
- **Prediction** is done from **left to right**
  - **start** with the example $x = x_1, ..., x_m$
  - assign the values to the **input nodes**
  - calculate the values of the **hidden nodes** of the next layer
  - **iterate** layer by layer until output
  - the **prediction** is in the **output** layer

Defining the topology: output

- **Binary classification**
  - 1 output node with **threshold**
  - 2 output nodes, choose maximum output
- **k class classification**
  - k output nodes, choose maximum
  - we can also produce a distribution using **softmax**
- **regression**
  - 1 output node, numerical value

Defining the topology: input

- **Number of input units**
  - One per numerical attribute
  - One per binary attribute
  - K-valued attribute
    - One per value (**one hot encoding**)

Defining the topology: layers

- **Number of layers**
  - **domain** dependent
  - as few as possible (**simplicity first**)
  - each layer **adds**
    - abstractive power (good)
    - overfitting risk (bad)
    - computational effort (bad)
- **Heuristic**
  - keep adding layers **until** learning stops improving
  - and **while** your resources allow
  - use cross-validation on the **validation** set
    - **not** on the **test set**

Defining the topology

- **Number of units** in hidden layers
  - no clear rules
- Some ideas
  - Low level input vars
    - **decrease** number of hidden nodes
  - High level input vars
    - **increase** number of hidden vars
  - Small data
    - **low number** of layers and units
- Trial and error
  - CV, ...
- Heuristic search, Genetic Algorithms, ...
- Meta learning
  - Learning how to learn

Learning

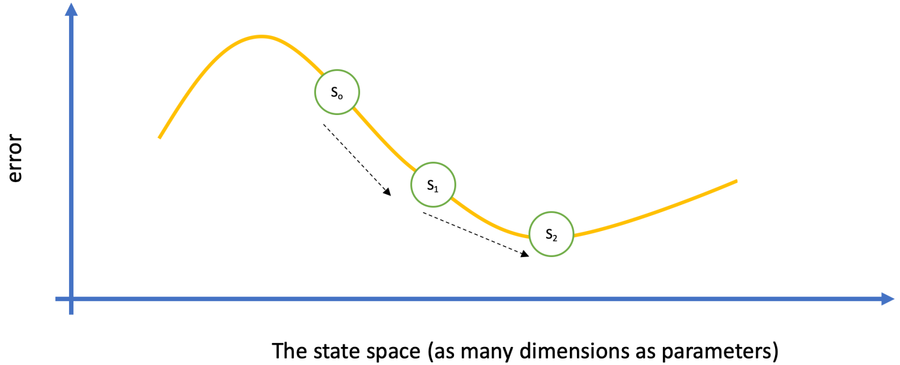
- How do we **train** a MLP?
  - algorithm **Backpropagation** (BP)
- BP:
  - **given**:
    - a set of examples
    - a network topology
  - **finds**
    - the “best” values for the weights (parameters)

Learning: Backpropagation

- **What** is the BP algorithm?
  - the **aim** is to reduce prediction error
  - **Init: start** the MLP with **random** weights
    - initial **state** of the NN
  - **Iterate:** update weights **optimally** according to observed errors
  - **until** convergence or maximum iterations

Learning: Backpropagation

- Backprogation
  - is an **optimization** algorithm
  - is derived **mathematically** from **first principles**



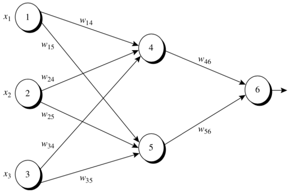
Backpropagation: in detail

- **Input**: D, topology, learning rate η
- **Output**: weights (trained model)
- **Do**
  - for each $x \in D$
    - calculate the outputs $\hat{o}$ using feedforward
    - calculate the derivative of error $err = derror(o, \hat{o})$ wrt weights
    - backpropagate *derror* from output to the first hidden layer
    - update the weights
  - until **stopping condition** is met (each iteration is an **epoch**)

Error calculation and propagation

- **Output derror** units calculated from output values and true values
  - assuming sigmoid activation, using the derivative of logistic

$$DErr_j = O_j(1 - O_j)(T_j - O_j)$$

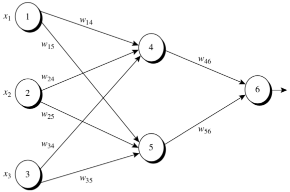


- **Example 9.1** (from Han et al.)
  - $x = (1, 0, 1)$, $o_6 = 0.474$, $T = 1$,
  - $DErr = (0.474)(1 - 0.474)(1 - 0.474) = 0.1311$

Error calculation and propagation

- **Hidden layer derror** units calculated from next layer k error

$$DErr_j = o_j(1 - o_j)\sum_k DErr_k w_{jk}$$

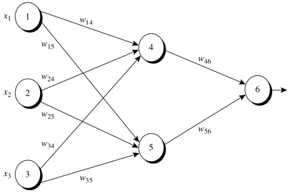


- Error of unit 4
  - $o_4 = 0.332,\ DErr_6 = 0.1311,\ w_{46} = -0.3$
  - $DErr_4 = 0.332(1 - 0.332)(0.1311)(-0.3) = -0.0087$

Updating weights

- Using error in firing unit

$$\Delta w_{ij} = \eta . DErr_j . o_i, \qquad w_{ij} = w_{ij} + \Delta w_{ij}$$



- $\eta = 0.9$, $DErr_4 = -0.0087$, (old) $w_{14} = 0.2$, $o_1 = x_1 = 1$
- $\Delta w_{ij} = (0.9)(-0.0087)(1) = -0.00783$
- $w_{14} = 0.2 + (-0.00783) = 0.19217$

Foundations of backpropagation

- Learning as optimization
  - objective is to minimize error or **loss**

$$\min_{w} L(y, \hat{y}) = \sum_{i=1}^{n} (y - \hat{y})^2$$

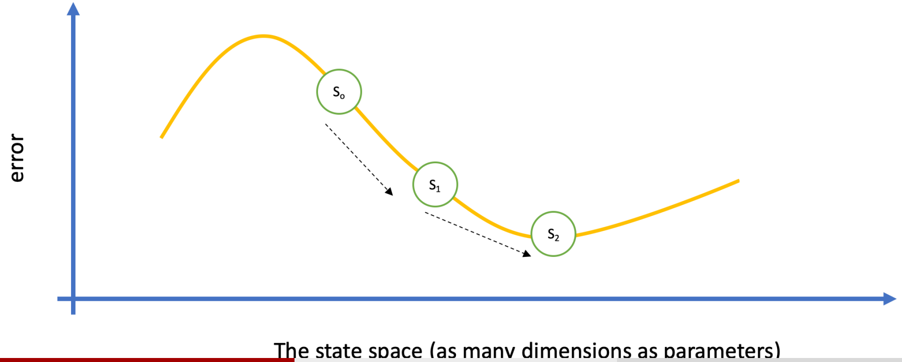
- We can define appropriate loss functions depending on the problem
- Solution is found by **deriving** E wrt parameters
  - no analytical solution for ANN (c.f. linear regression)

Stopping criteria

- The increments Δ_{ij} are too small
- Error is low enough
- Maximum number of iterations

Foundations of backpropagation

- We use an **iterative** approach based on **gradient descent**
  - steepest descent (descida mais rápida)
- backpropagation uses gradient descent
  - start from initial weights (random)
  - move in the space of solutions as indicated by the gradient
    - **learning rate** η is the size of the step



Limitations

- Local minima
  - BP is an **eager** algorithm
  - it may miss the global optimum
- Different scales of attributes
  - make learning take longer
  - usually we **normalize** input attributes

Limitations

- Random start
  - initial weights are **random**
  - usually Gaussian with mean zero
  - a constant would give the same initial output to all cases
- Variability
  - random start may lead to **different solutions** (local minima)
  - good idea to **repeat with different initializations** (no fixed seed)

Efficiency

- Given N cases and W weights
- Each **epoch** takes $O(N.W)$ operations
- The **number of epochs** depends on the data
  - easy problem converges quickly
- and on the number of weights
  - complex networks take longer to converge
  - limiting the number of iterations may be practical

Optimizers

- Backpropagation is the **classical** ANN optimizer
  - but there are many others
  - **Adam** is a popular one with Deep Learning
  - the most popular use gradient descent

Explainability

- ANN are **opaque** models
- We can read:
  - the coefficients of linear regression
  - the rules in a decision tree
  - the probabilities in Naive Bayes
- But the **weights** of a MLP are
  - not directly interpretable (no easy meaning)
  - combined to obtain an answer
- There are techniques for **obtaining explanations** from ANN models
  - **hot topic**
  - XAI: Explainable AI

References

- Books
  - Han, Kamber & Pei, Data Mining Concepts and Techniques, Morgan Kaufman.
- Wikipedia
  - Backpropagation, https://en.wikipedia.org/wiki/Backpropagation