A single neuron linear classifier

M. Salomon
FEMTO-ST Institute - DISC Department - AND Team
Univ. Bourgogne Franche-Comté (UBFC), France

January 16, 2024

1st year Master in Computer Science - Internet of Things

A linear neuron for classification

- Consider a problem with:
  - two inputs (x_1 and x_2)
  - one output (y^t)
- The function P_W implemented by the perceptron satisfies:
  - $P_W : \mathbb{R}^2 \to \mathbb{R}$
  - $P_W\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = f\left(w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2\right) = y$

  where $x_0 = +1$ is the *bias*
- When f is the identity function $f(x) = x$ (linear neuron) then
  - $P_W\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = w_0 \cdot 1 + w_1 \cdot x_1 + w_2 \cdot x_2 = w_0 + \sum_{i=1}^{2} w_i \cdot x_i$

Training of this linear perceptron

- Supervised case $\rightarrow$ a training set S
  - n observations which are input-output pairs
  - $S = \left\{\left(X_1, Y_1^t\right), \ldots, \left(X_n, Y_n^t\right)\right\}$, where $\left(X_j, Y_j^t\right)$ is a pair
- Supervised training $\rightarrow$ an optimization problem
  - Error / Loss function $L \rightarrow$ drives the training

$$\min_W \left[\frac{1}{n}\sum_{j=1}^{n} L\left(P_W(X_j), Y_j^t\right)\right]$$

$$\min_w \left[\frac{1}{n}\sum_{j=1}^{n} L\left(y_j, y_j^t\right)\right]$$

  - Mean Squared Error / Loss $\rightarrow L\left(y, y^t\right) = \frac{1}{2}\left(y - y^t\right)^2$
  - For a problem with two inputs and one output $\left(X_j, Y_j^t\right) = \left(\left(x_{1j}, x_{2j}\right), y_j^t\right)$ the objective function to be minimized is

$$\min_w \left[\frac{1}{n}\sum_{j=1}^{n}\frac{1}{2}\left(\left(w_0 + \sum_{i=1}^{2} w_i \cdot x_{ij}\right) - y_j^t\right)^2\right]$$

Training of this linear perceptron - 1/2

- Update rules / weights correction

$$w_i = w_i - \gamma \cdot \frac{\partial L}{\partial w_i}$$

  where γ is the *learning rate*

- Loss function seen as a composite function
  - $L(y_j, y_j^t) = \frac{1}{2} l_j^2$
  - $l_j = y_j - y_j^t$
  - $y_j = f(v_j) = v_j$ (f is the activation function $\rightarrow$ linear function)
  - $v_j = w_0 + \sum_{i=1}^{2} w_i \cdot x_{ij}$

Training of this linear perceptron - 2/2

- Gradient computation $\nabla L(y_j, y_j^t) = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}\right)^T$
- Computation of each component $\frac{\partial L(y_j, y_j^t)}{\partial w_i}$ using chain rule

$$\frac{\partial L(y_j, y_j^t)}{\partial w_i} = \frac{\partial L}{\partial l_j} \cdot \frac{\partial l_j}{\partial y_j} \cdot \frac{\partial y_j}{\partial v_j} \cdot \frac{\partial v_j}{\partial w_i}$$

where

- $\frac{\partial L}{\partial l_j} = l_j$; $\frac{\partial l_j}{\partial y_j} = 1$; $\frac{\partial y_j}{\partial v_j} = 1$
- $v_j = w_0 + \sum_{i=1}^{2} w_i \cdot x_{ij}$

Practical implementation - 1/2

- Let us consider a binary classification problem
  - Two classes (labels) $\rightarrow y_j^t \in \{0, 1\}$
  - Generate a dataset (`make_blobs`; `make_moons`)
    - Use `datasets` module of `sklearn`
    - Watch `Sample generators`
  - Display the data set with `mathplotlib`
- Write a function `predict` $\rightarrow$ computes the percep. output
  - *Inputs* $\rightarrow$ an observation from dataset; weights
  - *Output* $\rightarrow$ input vector class (label) y_j
    - 0 if $P_W(X_j)$ is negative
    - 1 if $P_W(X_j)$ is positive or zero
- Write a function `training` $\rightarrow$ trains the perceptron
  - *Inputs* $\rightarrow$ training set; learning rate; number of epochs
    (2 nested loops $\rightarrow$ *nb epochs*; *training set* then update w_i for $\left(X_j, Y_j^t\right)$)
  - *Output* $\rightarrow$ weights obtained after training

Practical implementation - 2/2

- Write a function `accuracy` → percep. classif. accuracy
  - *Inputs* → dataset; weights
  - *Output* → percentage of observations well-classified
- Write a function `crossValid` → cross-validation
  - *Inputs* → dataset; number of subsets (folds)
    (Split the data set in n disjoint testing subsets - *k-fold* with $k = n$)
    - ▶ Use `model_selection` module of `sklearn`

    learning rate; number of epochs
  - *Output* → vector with the weights of the best perceptron
- Combine the function to evaluate the performance of a perceptron using cross-validation validation on the data set
- Compare different variations of gradient descent (methods)
  - *Batch GD*, *SGD*, *Mini-batch GD*

Squared error and linear neuron as output → Regression pb

A sigmoid neuron for classification

- What is the drawback of a linear regression?
  - Observations far from the line separating the classes
    $\Rightarrow$ High impact on the position of the line
  - How to limit their influence?
    $\Rightarrow$ By transforming the values of the linear function

- Let us take as activation function the sigmoid one

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

- Sigmoid neuron output value is between 0 and 1
  - $P_W(X_j) < 0.5 \rightarrow y_j = 0$
  - $P_W(X_j) \geq 0.5 \rightarrow y_j = 1$

  The closest the output to 0.5, the more the result is unsure

Loss function bayesian interpretation

See the output of the sigmoid neuron as a probability

- Consider the output as the probability that $y_j^t = 1$
  - Hence $P(y_j^t = 1 | X_j; W) = P_W(X_j) = g(X_j)$
  - and thus $P(y_j^t = 0 | X_j; W) = 1 - P(y_j^t = 1 | X_j; W) = 1 - g(X_j)$
- Training process
  - What should it do?

    $\Rightarrow$ Penalize the more the probability y_j is far from y_j^t
  - Negative log-likelihood loss

    $\Rightarrow$ Squared error is difficult to optimize

$$
\begin{aligned}
L_{nlv}\left(y_j, y_j^t\right) &= -y_j^t \log\left(y_j\right) - \left(1 - y_j^t\right) \log\left(1 - y_j\right) \\
&= \begin{cases} -\log\left(1 - y_j\right) \text{ if } y_j^t = 0 \\ -\log\left(y_j\right) \text{ if } y_j^t = 1 \end{cases}
\end{aligned}
$$

Loss function bayesian interpretation

Taking negative of the log of Bernoulli Distribution

$$\begin{aligned} Binary\,Cross\,Entropy\,Loss(y,\hat{p}) &= -\log(P(y \mid x) \\ &= -(y\log(\hat{p}) + (1-y)\log(1-\hat{p})) \\ &= \mathbf{-y\log(\hat{p}) - (1-y)\log(1-\hat{p})} \end{aligned}$$