Machine Learning and Neural Networks

The theory behind implementations

Natanael Alpay and Dr. Mihaela Vajiac

Chapman University

CHAPMAN UNIVERSITY
SCHMID COLLEGE OF SCIENCE AND TECHNOLOGY

Outline

Neurons are the building blocks of the nervous system, that communicates with other cells via specialized connections called synapses. There are several different types of neurons that facilitate the transmission of information.

The perceptron is the building block of artificial neural networks and is an algorithm for supervised learning of binary classifiers function which decides whether or not an input, represented by a vector of numbers, belongs to some specific class. The perceptron is based on a simplified model of the biological neurons in our brain.

The perceptron algorithm was invented in 1958 by Frank Rosenblatt for linearly separable data. For non-linearly separable, we have RBF networks and support vector machine methods.

Neurons and History

The perceptron was first inspired by neurons which are the building blocks of the nervous system. Neurons receive and transmit signals to different parts of the body. This is carried out in both physical and electrical forms.

Perceptron

The perceptron is the building block for neural networks and is based on a simplified model of a single biological neuron in our brains, that imitate the real neuron by:

* Taking input layer - $Y = \langle y_1, \dots y_N \rangle$.
* Computer the weighted sum $\nu = \sum_i y_i w_i = \langle Y, w \rangle$.
* Pass it through the activation function activation function φ.
* Obtain the output layer if reach final layer, else pass result to followed hidden later.

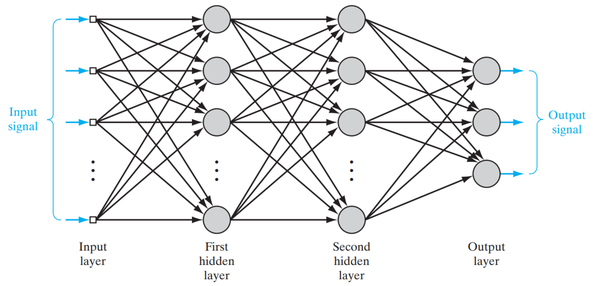

Figure 1: Multi-layer Perceptron with two hidden layers, from [1] .

The **decision boundary** is a hyper plane of dimension $n-1$ for n inputs, for 2 inputs, it is a line. The weights, w_i's, are perpendicular to the decision boundary and determines its slope.
Perceptron convergence theorem *states that there is a iterative way to find a decision boundary A which separate the two classes after a finite number of iteration.*

Perceptron Theorem *states that a single layer system can be used to describe any Boolean function using the activation function.*

Activation function

A continuous function φ whose derivative is used to calculate δ, hence its only restriction is to be differentiable, such as:

Logistic function: of the form

$$\varphi_j(v_j(n)) = \frac{1}{1+\exp(-av_j(n))}, \quad a > 0.$$

with derivative

$$\varphi_j'(v_j(n)) = \frac{a\exp(-av_j(n))}{[1+\exp(-av_j(n))]^2} = ay_j(n)[10y_j(n)]$$

Binomial function: of the form

$$\varphi_j(v_j(n)) = \begin{cases} 0, & , v_j(n) < 0 \\ 1, & , v_j(n) \geq 0 \end{cases}.$$

Perceptron Algorithm

The **perceptron algorithm** was invented in 1958 by Frank Rosenblatt, and is an iterative algorithm to find the decision boundary where $D = \{d_1, \dots, d_N\}$, $d_i = \pm 1$ is the target/desired result type.

We update based on the perceptron prediction $x \cdot w$, for a current weight. Since $\varphi(\nu) \leq 0$ if and only if the predicted label is different from d, which in that case we update the weight by $w = w + yx$.

This choice of update is to bring our current wrong prediction closer to the current positive value, and if the data is linearly separable (as we require it to be), by updating each point for a certain number of times, the weights will eventually converge to a current boundary.

Perceptron Algorithm Implementation

Following the perceptron algorithm, we implement the weight update for a dataset of the form $\tau = \{(y_i, d_i)\}$, as follows

```
For i in range of (0, 'num of iterations')
  For each pair (y, d) in (Y, D):
    ν = d(y · w)
    If φ(ν) ≤ 0 :
      w = w + yx    -we update the weight
```

When implementing the perceptron we first split our data set, and *train* it on a part of our data set, that we perform the perceptron for the first time. Then to *test*, we perform the perceptron on the other part of the data set.

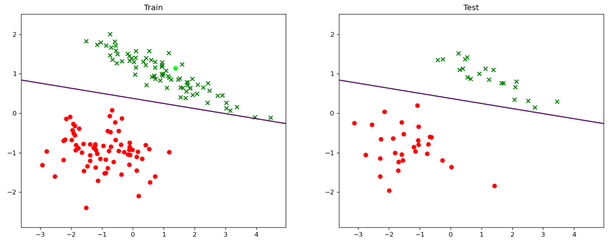

Figure 2: Perceptron Algorithm implemented using binomial activation function. The train set on the left and test set on the right, this allows us to test our classifier with other points to verify it works not only for our dataset.

Radial Basis function

The **radial-basis function (RBF)** classification is used on the cases when data is not linearly separable. We move the problem to higher dimension to separate the classes of the output layer.

Micchelli's Theorem *states that for $\{x_i\}_{i=1}^N$ set of distinct points in $\mathbb{R}^{m_0}$. The $N \times N$ interpolation matrix $\Phi = \{\varphi_{i,j}\}_{i,j=1}^N$ i.e. ji-th element $\varphi_{ij} = \varphi(||x_i - x_j||)$, is non singular.*

The Gaussian positive definite radial basis function kernel is

$$\varphi_j(x_i, x_j) = \exp\left(-\frac{1}{2\sigma_j^2}||x_i - x_j||^2\right) \leq 1,$$

with $\sigma_j^2 = \sum_{C(i)=j} ||x_i - \hat{\mu}_j||^2$ for encoder $C(i) = j$ that maps x_i to jth cluster of data,

With the **K-mean** algorithm we find σ by minimizing the **cost function** $J(C) = \sum_{j=1}^K \sigma_j^2$. Then the **recursive least squares** (RLS) minimizes $J(c)$ with new weight each iteration.

Recursive Least Squares Algorithm

The **least square form** is $\hat{w}(n) = r(n)R^{-1}(n)$, where

1. Define recursively K-to-K correlation function $R(n)$ as

$$R(n) = \sum_{i=1}^n \Phi(x_i)\Phi^T(x_i) = R(n-1) + \Phi(n)\Phi^T(n).$$

2. Define encoder between desired output and hidden output

$$r(n) = \sum_{i=1}^n \Phi(x_i)d(i).$$

3. Weight vector $\hat{w}(n)$ that is optimized by least-square method.

Consider the linear regression model $d(n) = w^T\Phi(n) + \epsilon(n)$, where $\epsilon(n)$ is error term of zero variance σ_ϵ^2. Then with the state-error covariance matrix $\mathbb{E}[(w - \hat{w}(n))(w - \hat{w}(n))^T] = \sigma_\epsilon^2 R^{-1}(n)$.

We get the final simplification for the RLS using the RLS **gain vector** $g(n) = R^{-1}(n)\Phi(n)$ and get the **updated weight** as

$$\hat{w}(n) = \hat{w}(n-1) + g(n)\alpha(n),$$

using the **prior estimation error** before the update

$$\alpha(n) = d(n) - \Phi^T(n)w(n-1) = d(n) - w^T(n-1)\Phi(n).$$

Key points to note:

1. The maximum value that the RBF kernel φ_j can be is 1 and happens when $||x_i - x_j||^2 = 0$, i.e. points are the same $x_i = x_j$.
2. Distance $||x_i - x_j||^2$ represents an **analog** to dissimilarity, as if distance between the points increases, less similar, i.e. $\varphi_j < 1$.

RBF Algorithm and Implementation

Using the K-mean and RLS Algorithm we can describe a hybrid learning procedure for a RBF network, whose advantage is its computational effectiveness. Works as follows:

Input layer, Take an input vector x denoted by m_0.
Hidden layer, Compute σ_j for φ_j using the K-mean algorithm.
Output layer, Get output layer $\Phi(x_i)$ and perform RLS algorithm.
Repeat , Perform algorithm until all $\varphi_j \approx 1$.

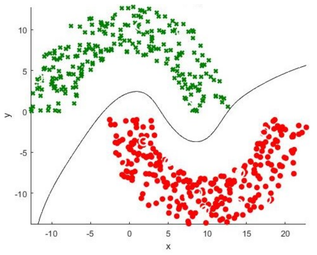

Figure 3: RBF hybrid learning algorithm implementation

Support vector machine

Support vector machine is an algorithm that constructs a hyperplane such that the margin of separation is maximized. Given data set $\{x_i, d_i\}$, we define the optimum hyper plane as

$$w^T \cdot \Phi^T(x) = \sum_{j=1}^\infty w_j\varphi_j(x) \begin{cases} \geq 0, d_i = +1 \\ \leq 0, d_i = -1 \end{cases}$$

with weight vector defined as $w = \sum_{i=1}^N \alpha_i d_i \Phi(x_i)$ for Lagrange multipliers $\alpha_i \geq 0$ such that $\sum_{i=1}^N \alpha_i d_i = 0$.
Support vectors are the vectors in which the hyper plane equation is 0, we define the **Kernel** using the inner product

$$K(x, x_i) = \Phi^T(x_i)\Phi^T(x) = \sum_{j=1}^\infty \varphi_j(x_i)\varphi_j(x).$$

So, $w^T\Phi(x)^T = \sum \alpha_i d_i \Phi^T(x_i)\Phi(x) = \sum \alpha_i d_i K(x, x_i) \geq 0$.
Mercer's Theorems *states that a symmetric Kernel K defined in closed interval can be extended in the series form*

$$K(x, x') = \sum_{i=1}^\infty \lambda_i \varphi_i(x)\varphi_i(x'), \lambda_i > 0$$

and it would converge absolutely iff $\int_a^b \int_a^a K(x, x')\psi(x')dxdx' \geq 0$ for all $\int_a^b \psi^2(x)dx < \infty$.
The kernel allows the construction of surface that is linear the image space. Define the dual form for **SVM optimization**:

$$Q(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j d_i d_j K(x_j, x_i),$$

the only requirement is for K to satisfy Mercer's Theorem, such as the Gaussian RBF.

References

[1] Haykin, Simon S.. Neural networks and learning machines. Third Upper Saddle River, NJ: Pearson Education, 2009.

[2] Minsky, M. and S. Papert. "Perceptrons: An Introduction to Computational Geometry, Expanded Edition." (1987).