Principles of Machine Learning

Unit 2

Supervised Learning

Neural Networks and Linear Discriminants: Brain and the neuron - neural networks - perceptron - linear separability - linear regression; multi-layer perceptron - forward and backward propagation; support vector machines.

* **Neural Networks** - an interconnected group of neurons

  eg. human brain

* **Artificial Neural Network** - a mathematical or computational model for information processing based on a connectionist approach to computation

* **Biological Neurons** - A neuron collects inputs from other neurons using dendrites.

  → The neuron sums all the inputs and if the resulting value is greater than a threshold, it fired.

  → The fired signal is then sent to other connected neurons through the axon.

* **Artificial Neurons** - **McCulloch & Pitts Neurons**

  → greatly simplified biological neurons

  → properties are derived simply by adding up the weighted sum as its input.

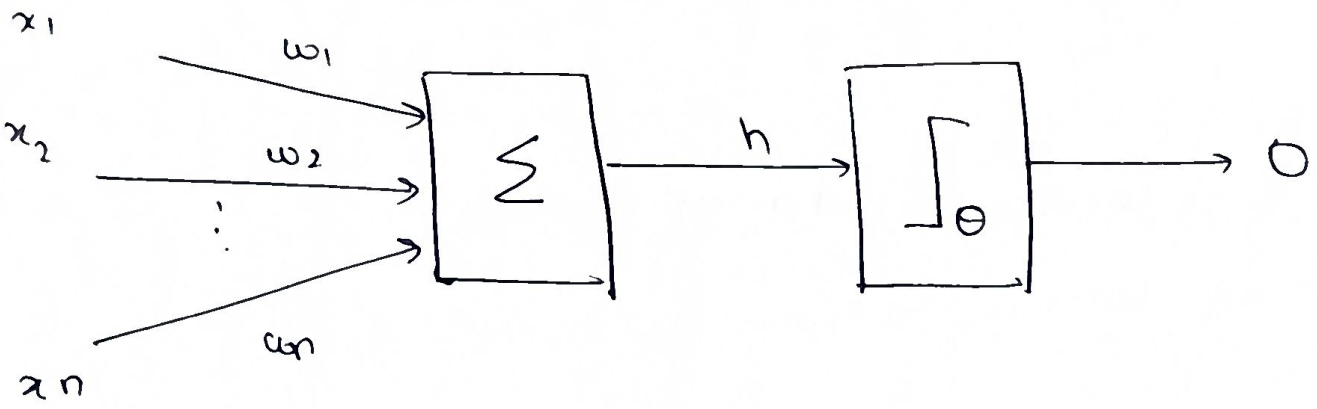
→ Aprocess output is derived using logical circuits if the weighted sum is greater than a threshold.

→ can be represented as:

$$h = \sum_{i=1}^{m} w_i x_i \qquad O = \begin{cases} 1, & h \geq \Theta \\ 0, & h < \Theta \end{cases}$$

Θ = threshold

→ The MP neuron is:



The components are:

(i) a set of weighted inputs that correspond to the synapses

(ii) an adder that sums the inputs

(iii) an activation function (threshold) that describes whether the neuron spikes or not

Hebb's Rule

→ Hebb's rule states that the changes in the strength of the synaptic connections are proportional to the correlation in the firing of the 2 neurons.

For eg. 1. Show food to dog
2. Ring bell

→ The neurons for salivating over food and hearing the bell fired simultaneously become strongly connected

→ Over time, hearing the bell will cause salivating.

Hebb's law can be represented in the form of two rules :

1. If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.

2. If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.

* Limitations of MP Neuron

→ The inputs to a real neuron are not necessarily summed linearly - there may be non-linear summations.

→ Real neurons do not output a single response, but rather a spike train. (a sequence of pulse)

→ The firing threshold changes over time in biologically neurons.

→ The neurons are not updated sequentially according to a clock, but rather, randomly (asynchronous)

→ The weights can change from +ve to -ve or vice versa - which has not been seen biologically - synaptic connections are either excitatory or inhibitory - they never change from one to another.

→ Real neurons can have synapses that link back to themselves in a feedback loop, but that is not done w/ the MP neuron

* Neural Network

→ The arrangement of neurons into layers and the connection patterns within and between layers is called the net architecture.

→ Can be built by putting a lot of McCulloch & Pitts neurons together.

→ Neural networks structures are classified into two catogories based on the connections established in the network. Those are:

A. **Feed Forward Networks**

→ represents any arbitrary function, no internal states.

includes

- single layer perceptron
- multi layer perceptron

B. **Recurrent Networks**

→ Output is fed back with delay, has internal states and can oscillate: includes

- Hopfield Network
- Boltzmann Machine

* Perceptron learning Algorithm

→ The perceptron is a collection of MP neurals together with a set of inputs and weights to fasten the inputs to the neurals

→ It has the following components:

(i) Inputs - input vector to the algo (x_i $i = 1$ to m)

(ii) weights - weighted connections between i and j (in matrix w)

(iii) Outputs - y_j ($j = 1$ to n dimensions)

The output can be expressed as $y(x, w)$

(iv) Target - t_j - the correct labels, for supervised learning

(v) Activation Function - $g(\cdot)$

(vi) Error - E - difference between y and t

Complexity = $O(Tmn)$
T iterations, n nested loop

* Learning Rate - η

→ The value of the learning rate decides how fast the network learns. It influences the extent to which the weights should be change.

High Learning Rate :- weights can change a lot whenever there is a wrong answer

- tends to make the network unstable, so that it never settles down.

Small Learning Rate :- the weights need to see the inputs more often before they change significantly, so that the network takes longer to learn.

- However, it will be more stable and resistant to noise (errors) and inaccuracies

Ideal learning rate: $0.1 < \eta < 0.4$

* Bias

If

→ All of the inputs to a neuron are zero, no matter what weights are set, the neurons would to the same thing, since the thresholds are fixed.

→ An extra input weight is added, usually -1, but any non-zero value can be used.

* Perceptron Algorithm

Initialization

- set all of the weights w_{ij} to small (+ve and -ve) random numbers

Training

- for T iterations or until all the outputs are correct

for each input vector

(i) compute the activation of each neuron j using the activation function g:

$$y_i = g\left(\sum_{i=0}^{m} w_{ij} x_i\right) = \begin{cases} 1 \text{ if } \sum_{i=0}^{m} w_{ij} x_i > 0 \\ 0 \text{ if } \sum_{i=0}^{m} w_{ij} x_i \leq 0 \end{cases}$$

(ii) update the weights individually using:

$$w_{ij} \leftarrow w_{ij} - \eta (y_j - t_j) \cdot x_i$$

Recall

Compute the activation of each neuron j using:

$$y_j = g\left(\sum_{i=0}^{n} w_{ij} x_i\right) = \begin{cases} 1 \text{ if } w_{ij} x_i > 0 \\ 0 \text{ if } w_{ij} x_i \leq 0 \end{cases}$$

* Activation Functions

1. Binary Step Function

$$y_i = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_{ij} x_i > 0 \\ 0 & \text{if } \sum_{i=0}^{m} w_{ij} x_i \leq 0 \end{cases}$$

2. Signum Function

$$y_i = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases}$$

3. Sigmoid Function : $y_i = \frac{1}{1+e^{-x}}$

* Limitations of Perceptron

→ neurons are completely independent of each other.

→ used mainly only for binary classification

→ can be used to classify linearly separable sets of input vectors.

* Linear Separability

→ Linear separability refers to the ability to separate data points of different classes using a linear decision boundary.

→ If linearly separable, then it is possible to draw a straight line, plane or hyperplane, separating the data points of classes

→ In a perceptron model, the line / hyperplane divides where the neuron fires on one side, and doesn't on the other.

→ This line is called the decision boundary or discriminant function

Geometry of Linear Separability

→ Consider a line in two dimensional space.

$$w_0 + w_1 x + w_2 y = 0$$

(x, y) are the coordinates of a point on the line.

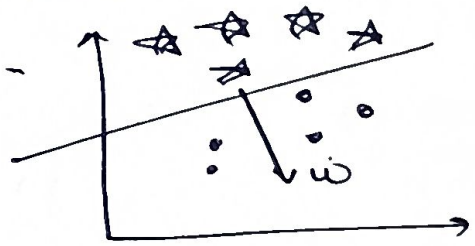
w_0, w_1, w_2 are the parameters of the line equation.

→ In vector form, the equation would be

$$w \cdot x = 0$$

$$w = (w_0, w_1, w_2)$$

$$x = (1, x, y)$$

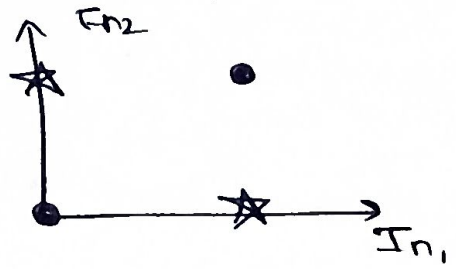


→ When the dot product $w \cdot x > 0$, it implies that the angle between vectors w and x is less than 90 degrees.

→ If the angle is acute, it means that the points lie on the same side of the line $w \cdot x = 0$

Example

→ The XOR Function, can be solved by a perceptron since it is not linearly separable



→ It can be solved in a 3D plane instead.

→ It is always possible to separate out two classes with a linear function, provided that the data is projected onto the correct set of dimensions.

→ Kernel classifiers, which form the basis for SVM, can deal with non-linearly separable data.

* Linear Regression

Regression - learn a function estimator $f: x \rightarrow R$, from data points $(x_i, f(x_i))$

The difference between the actual and estimated function values are called residues $E_i = f(x_i) - f'(x_i)$

Linear Regression

→ Consider the equation of a line: $y = a + bx$

b = slope

a = y-intercept.

→ If there is a dataset which is strongly correlated and exhibits a linear relation, the principle of least squares can be used to draw a line through the dataset, such that the sum of the squares of the deviations of all points from the line is minimized.

→ In a simple linear regression, predict scores on one variable from the value of another variable.

Variable being predicted = criterion variable = Y

Variable on which predictions are made = Predictor variable = X

→ For each point on the dataset :

$y - (a+bx)$ = vertical distance from line to point

→ Compute $\sum [y - (a+bx)]^2$ ⇒ The line that minimizes the sum of the squard distance is the Least Squares Line

$y = \beta_0 + \beta_1 x$

$\beta_1 = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sum (x-\bar{x})^2}$

$\bar{y} = \beta_0 + \beta_1 \bar{x}$

or → $\beta_1 = \frac{n\sum xy - \sum x \sum y}{n(\sum x^2) - (\sum x)^2}$

* Multi-Layer Perceptron

→ A MLP uses a backpropagation network. A backpropagation network can be trained in 3 stages :

(i) feed forward of the input training pattern

(ii) back propagation of the associated error

(iii) adjustment of weights.

Algorithm

→ Each training example is in the form of (x,t) where x is the input vector and t is the target vector.

→ η is the learning

→ n_i = network inputs

→ n_{hidden} = no. of hidden layer units

→ n_{out} = no. of output units

→ The input from i to j is x_{ji} and weight is w_{ji}

1. Create a feed forward network with ni inputs, n_{hidden} and nout

2. Initialize the network weights to small random numbers

3. Until the termination condition is met - Do:

(i.) Input the instance x to the network & compute the output o_u of each unit u.

(ii) For each network unit k, calculate the error δ_k

$$\delta_k \leftarrow O_k(1-O_k)(t_k - O_k)$$

(iii) For each network unit h calculate the error δ_h

$$\delta_h \leftarrow O_h(1-O_h) \sum_{k \in outputs} w_{hk}\delta_k$$

(iv) Update the network weights w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

* Support Vector Machines

→ The primary objective of SVM is to find the optimal hyperplane that best separates the data points into different classes.

→ SVM works as follow:

① Find hyperplane - The hyperplane that best separates the data points into different classes is identified.

→ The hyperplane is selected on the basis of the maximum margin, which is the distance between the hyperplane and the nearest data points from each class, called the support vectors.

② Kernel Trick :- used to handle non-linearly separable data with the kernel trick.

→ The kernel function transforms the input data into a higher dimensional space, where it can be linearly separated

→ Some common Kernel functions are:

(i) linear
(ii) polynomial
(iii) Radial Basis Function
(iv) Sigmoid Kernels

③ Margin Maximization - SVM aims to maximize the margin between the support vectors of different classes, which helps improve generalization & makes the model less sensitive to noise.

④ Regularization Parameters - The regularization parameter C controls the tradeoff between maximizing the margin & minimizing the classification error.

Higher C ⇒ more complex decision boundaries ⇒ overfitting

lower C ⇒ larger margin ⇒ may underfit

* Derivation of Decision Boundary for SVM

Hard Margin SVM — The algorithm aims to find the hyperplane that perfectly separates the classes without allowing any misclassifications.

The equations are:

For $y = +1$ $\quad w^T x_i + b \geq 1$

For $y = -1$ $\quad w^T x_i + b \leq -1$

$$\Downarrow$$

$$\boxed{y_i \cdot (w^T x_i + b) \geq 1}$$

$$\Rightarrow 1 - y_i (w^T x_i + b) \leq 0$$

The optimization problem is:

$$\min_{w,b} \left(\frac{1}{2} \|w\|^2 \right) \quad \text{subject to} \quad y_i (w^T x_i + b) \geq 1$$

Soft Margin SVM — The algorithm allows for some misclassifications, this is used when the data is not perfectly separable.

→ The soft margin SVM introduces a slack variable ξ_i for each data point, representing the deviation from the right side of the hyperplane.

The optimization problem is

$$\min_{w,b,\xi} \left(\frac{1}{2} \|w\|^2 \right) + C \sum_{i=1}^{n} \xi$$

subject to $y_i(w^T x_i + b) \geq 1 - \xi$ & $\xi \geq 0$

C is the regularization parameter.

This optimization problem can be solved using Lagranaian Multipliers.

The Lagranaian L for the problem is

$$L(w^T, b, \alpha) = \frac{1}{2}\|w^*\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i (w^{*T} \cdot x_i + b) - 1 \right]$$

$$= \frac{1}{2} w^T \cdot w - \sum_{i=1}^{n} \alpha_i \left[y_i (w^T x_i + b) - 1 \right]$$

The KKT condition is:

$$= \frac{1}{2} \sum_{k=1}^{m} w^k w^k + \sum_{i=1}^{n} \alpha_i \left[1 - y_i \left(\sum_{k=1}^{m} w^k x_i^k + b \right) \right]$$

n = no. of examples

m = dimension of the space

Differentiate w.r.t w and b, and set to 0.

(↖ $w^k w^k$)

w.r.t L $\frac{1}{2}\|w\|^2$ w.r.t w $= w$

$$\sum_{i=1}^{n} \alpha_i \left(1 - y_i \left(\sum_{k=1}^{m} w^k x_i^k + b \right) \right) \text{ w.r.t } w$$

$$= \sum_{i=1}^{n} \alpha_i (-y_i) x_i = 0$$

$$\therefore \frac{\partial L}{\partial w^k} = w_i - \sum_{i=1}^{n} \alpha_i (-y_i) x_i$$

$$= 0$$

*

w.r.t B $\quad \frac{1}{2}\|w^2\|$ w.r.t $b=0$

$$\sum_{i=1}^{n} \alpha_i \left(1 - y_i \left(\sum_{k=1}^{m} w^k x_i^k + b\right)\right) \quad \text{w.r.t } b$$

$$= -\sum_{i=1}^{n} \alpha_i y_i$$

$$\therefore \frac{\partial L}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i$$

$$\frac{\partial L}{\partial w^k} = w + \sum_{i=1}^{n} \alpha_i (-y_i) x_i = 0$$

$$\Rightarrow \boxed{w = \sum_{i=1}^{n} \alpha_i y_i x_i}$$

Substitute w in the Lagrangian equation:

$$L = \frac{1}{2} \sum_{i=1}^{n} \alpha_i y_i x_i^T \cdot \sum_{j=1}^{n} \alpha_j y_j x_j + \sum_{i=1}^{n} \alpha_i \left[1 - y_i \left(\sum_{j=1}^{n} \alpha_j y_j x_j^T x_i + b\right)\right]$$

$$= \frac{1}{2} \sum_{i=1}^{n} \alpha_i y_i x_i^T \cdot \sum_{j=1}^{n} \alpha_j y_j x_j + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i y_i \sum_{j=1}^{n} \alpha_j y_j x_j^T x_i$$

$$- b \sum_{i=1}^{n} \alpha_i y_i \quad (\to 0)$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\therefore L = \frac{1}{2} \sum_{i=1}^{n} \alpha_i y_i x_i^T \cdot \sum_{j=1}^{n} \alpha_j y_j x_j + \sum_{i=1}^{n} \alpha_i$$

→ This function is in terms of only α_i. According to the dual problem, if w is known, α_i is known and vice versa.

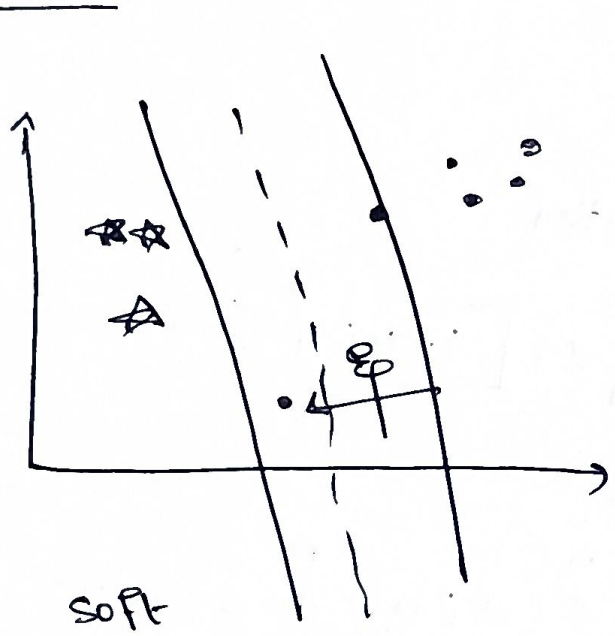
→ The objective fn. of the dual function needs to be maximized

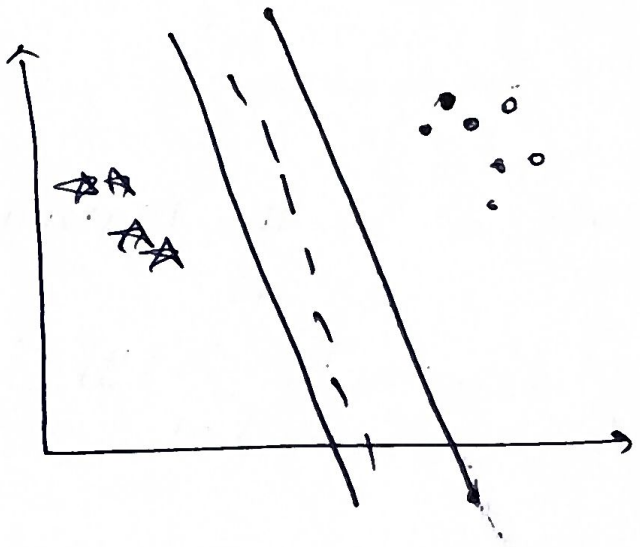
$$\max \omega(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

Subject to $\alpha_i \geq 0$ $\sum_{i=1}^{n} \alpha_i y = 0$ ⟶ for hard margin

for soft margin $\frac{2}{\|w\|} + \frac{C}{N} \sum_{i=1}^{n} \xi_i$

Graphs





Hard Margin vs. Soft Margin

* Examples of Kernel Functions

(1) Polynomical Kernel - $K(x,y) = (x^T y + 1)^q$

(2) Radial-based function - $\exp(-\|x-y\|^2 / 2\sigma^2)$

(3) Sigmoid $= \tanh(kx^T y + \theta)^n$