Lecture 18

Name: Harsh Sanjay Roniyar

Roll Number: 22B3942

Continuing from last lecture, these are the steps that we follow -

1. Change objective to a Lagrangian
2. Equate derivatives wrt w, b to 0
3. Substitute
4. Manipulate the loss equation
5. Rewrite the constraints as KKT Conditions

We performed the above steps in Lecture 17.

For Soft SVM, we got box constraints for the result. Classification with respect to that is as follows -

Box constraints: Four types of points

- **Inside the margin (and correctly classified)**
  - $a_n = 0$
- **On the margin (and correctly classified)**
  - $0 < a_n < C,\ \xi_n = 0$
- **Across the margin and correctly classified**
  - $a_n = C,\ \xi_n \leq 1$
- **Across the margin and misclassified**
  - $a_n = C,\ \xi_n > 1$

For Hard SVM, we will have $C = \infty$. (`More regularization means smaller C`)

The hyperparameters in this setting are -

1. . C
2. Kernel hyperparameters (K and d)

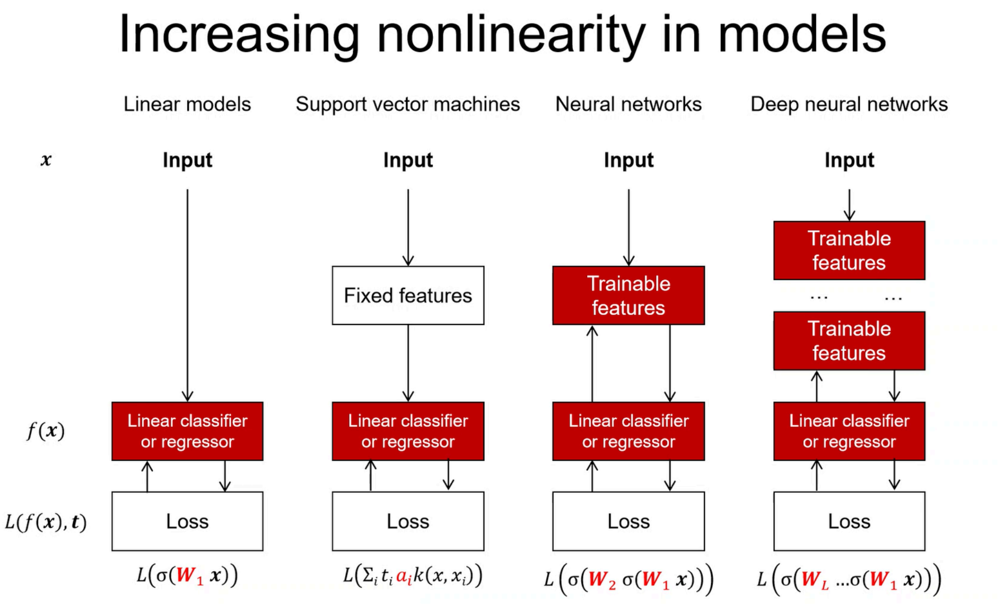
Thus, we conclude the Kernel Methods, Kernelized SVMs.

We now move to begin Neural Networks

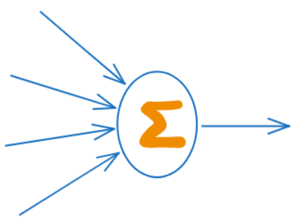
Introduction to Neural Networks

key terms : `hidden layers`, `mathematical modelling`

A very informative representation of all models in order of increasing non-linearity.



Tries to mimic the behavior of a biological neuron.



Layered structure of neuron -

$$y = w_3^{Q\times P}\sigma\left(w_2^{P\times M}\sigma\left(w_1^{M\times N}\mathbf{w}^{N\times 1}\right)\right)$$

where w_1, w_2, w_3 represent the weights for the layers 1, 2 and 3 respectively and σ is a non-linear function known as the activation function (since we want to perform non-linear modelling).

Application of chain rule for gradient update/descent is known as back-propagation.

Importance of hidden layers

The hidden layers iteratively extract the features from the previous stage of features until the desired output stage is reached.

$$f(x) = \sum_{i\in S} w_i k(x, x_i)$$

Universal Approximation Theorem

The Universal Approximation Theorem states that a neural network with at least one hidden layer of a sufficient number of neurons, and a non-linear activation function can approximate any (`Lipschitz`) continuous function to an arbitrary level of accuracy, with the approximation error controlled by the Lipschitz constant of the function being approximated.

Types of activation functions (σ)

- **Step**
- **Sigmoid**
- **ReLU** (Rectified Linear Unit)
- **Softmax**
- **tanh**
- **Linear**

> Formally, the standard (unit) softmax function $\sigma : \mathbb{R}^K \to (0,1)^K, \text{ where } K \geq 1,$ takes a vector $\mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$ and computes each component of vector $\sigma(\mathbf{z}) \in (0,1)^K$ with
>
> $$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax Function - Wikipedia.

`PS: Refer to this` Neural Networks - Harsh S Roniyar `amazing link for more such content`