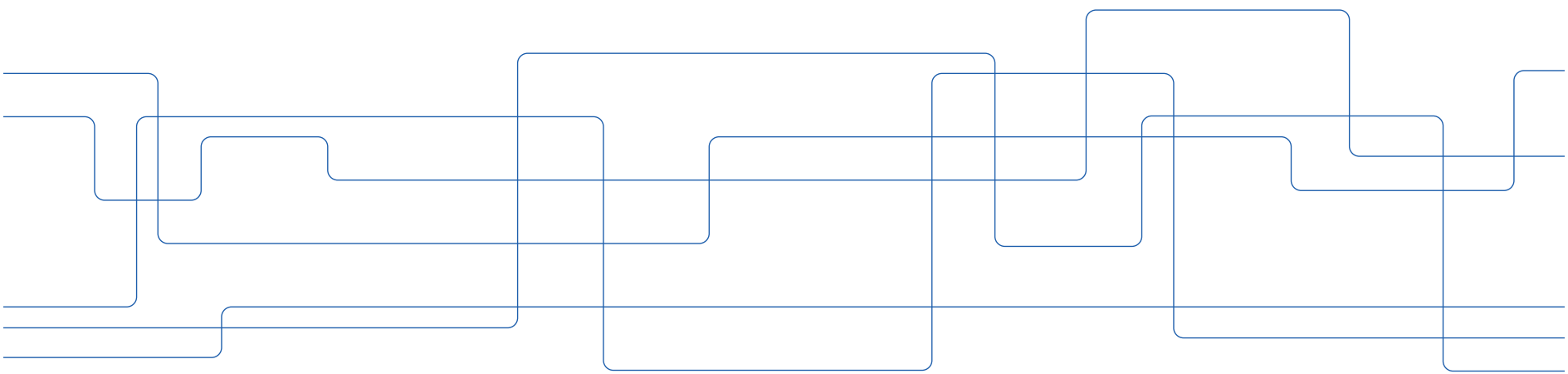
ID2214/FID3214
Programming for Data Science
– Neural Networks

Niharika Gauraha

Lecturer
Division of Software and Computer Systems
Department of Computer Science
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
niharika@kth.se



Outline

Perceptron

Basis Functions

Activation Functions

Neural Network

Perceptron

Perceptron is a linear binary classifier.

- Given training data $\{(X_i, y_i)\}$ for $i = 1, \ldots, n$, with $X \in \mathbb{R}^p$ and $y_i \in \{-1, +1\}$,
- Hypothesis:

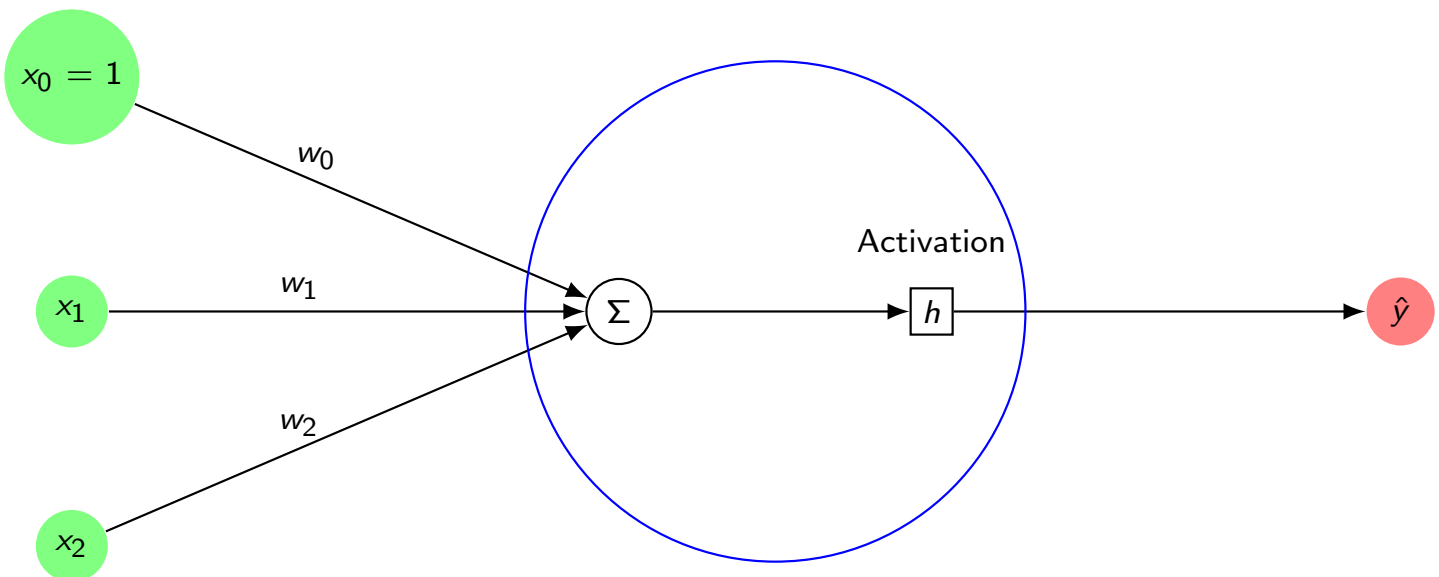
$$f(x) = \begin{cases} \geq 0, & y = 1 \\ < 0, & y = -1 \end{cases}$$

- The goal is to minimize the classification error.

Perceptron

Perceptron consists of fours parts:

- Input values X
- Weights and Bias $\boldsymbol{w}$
- Weighted sum of input $\boldsymbol{w}^T X$
- Activation Function $f(x) = \begin{cases} \geq 0, & y = 1 \\ < 0, & y = -1 \end{cases}$



Perceptron: Algorithm

Algorithm 1 Perceptron Algorithm

$\boldsymbol{w} = 0$ Initialize weight vector with 0
for *iter* $= 1$ to *num_iterations* **do**
  for $i = 1$ to n **do**
    if $\boldsymbol{w}^T X_i > 0$ and $y_i = -1$ **then**
      $\boldsymbol{w} = \boldsymbol{w} - X_i$
    else if $\boldsymbol{w}^T X_i < 0$ and $y_i = 1$ **then**
      $\boldsymbol{w} = \boldsymbol{w} + X_i$
    end if
  end for
end for

It is also known as Single Layer Perceptron.

Perceptron Example

Linear Separable case:

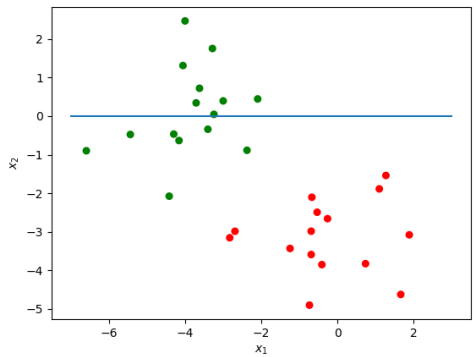


Figure: Classifier with initial weight

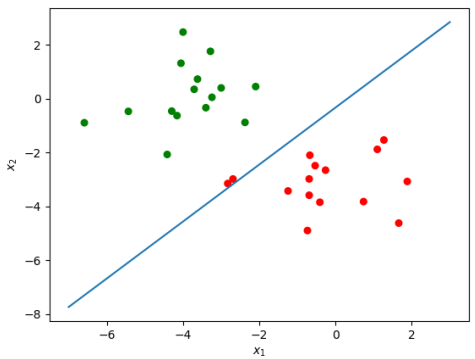


Figure: Classifier after iteration 1

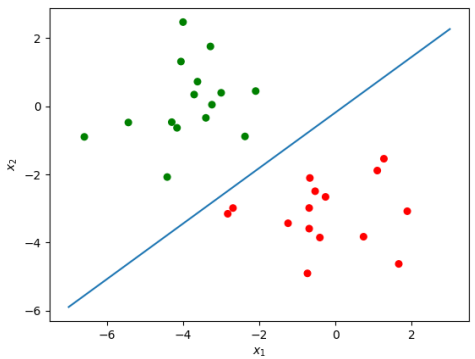
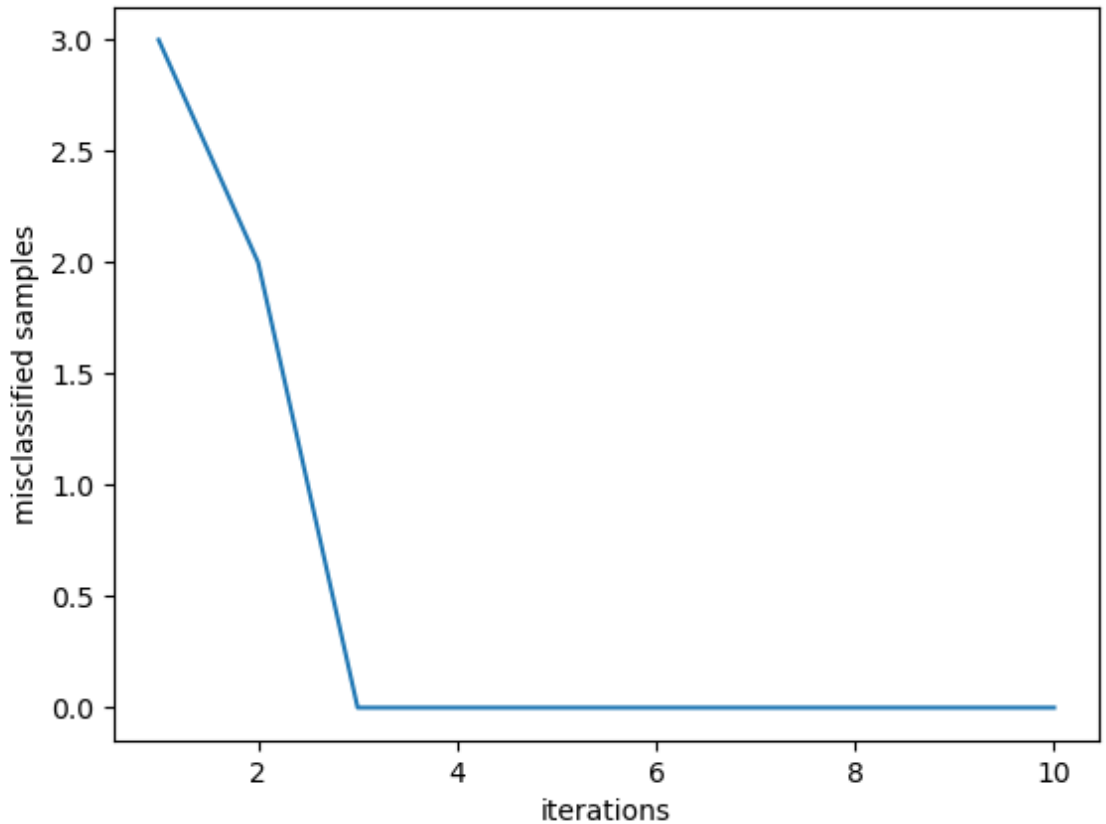


Figure: Classifier after iteration 2

Perceptron Example

If the data is linearly separable then Perceptron converges after a finite number of iterations.



Perceptron Example: Non linearly separable case

Non Linearly Separable case: Perception does not converge

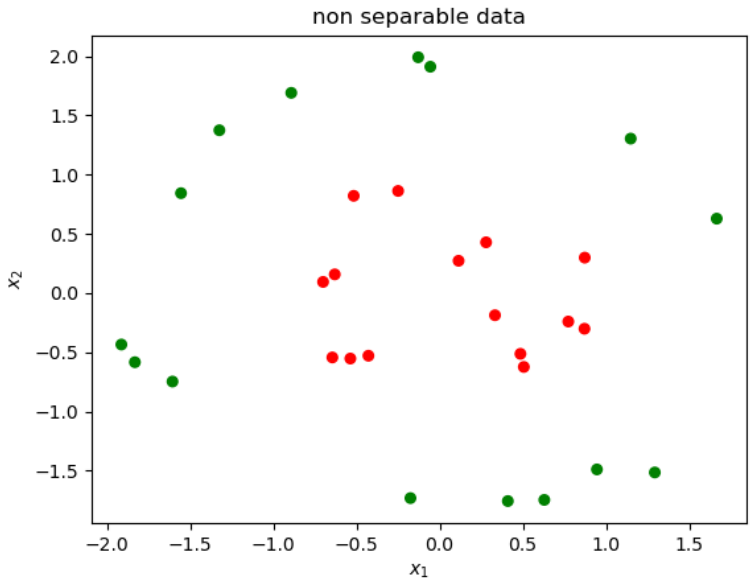


Figure: Non linearly separable data

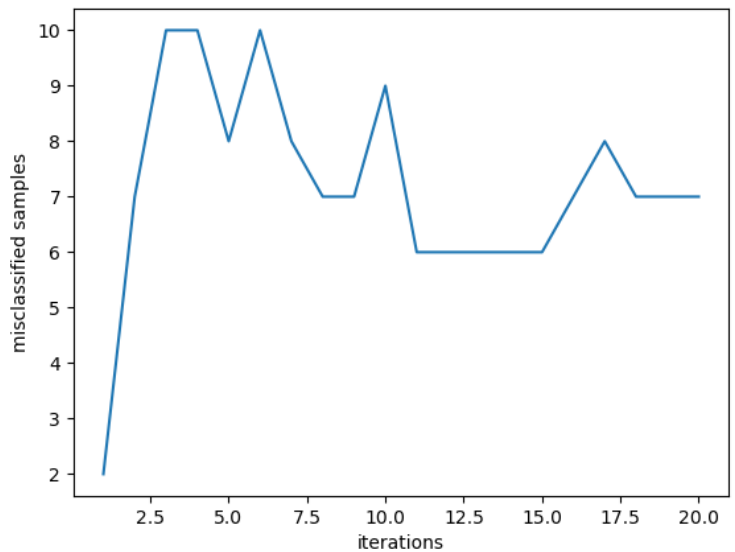


Figure: Iterations vs number of missclassified data points

Polar Coordinates

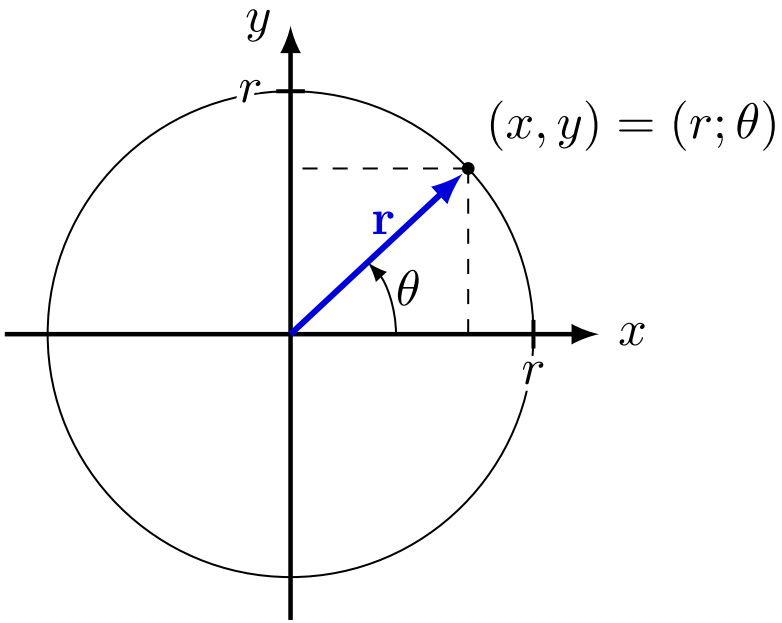


Figure: Polar Coordinates

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \arctan \frac{y}{x}$$

Perceptron Example: Transformed Space

Transformed into polar coordinates:

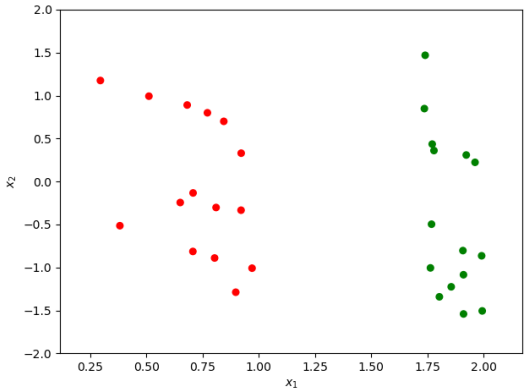


Figure: iteration 0

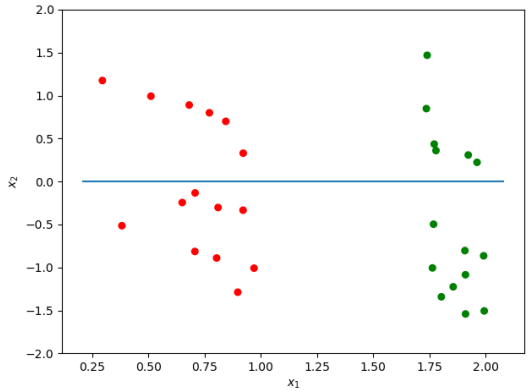


Figure: iteration 1

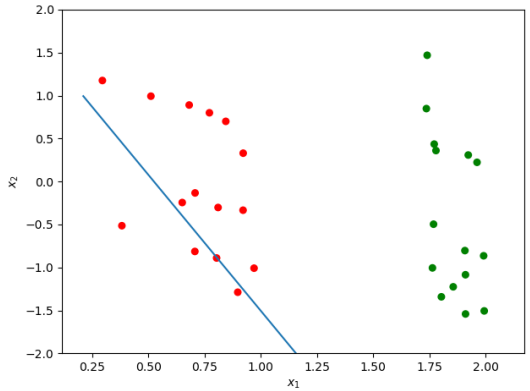


Figure: iteration 2

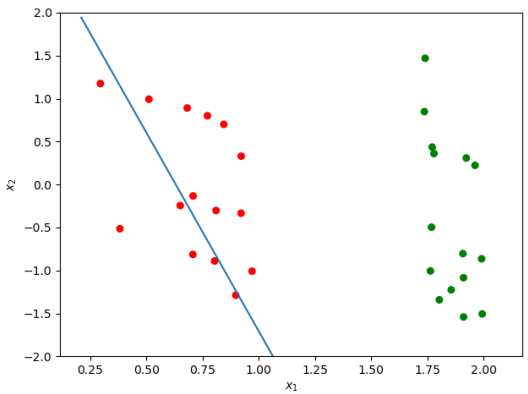


Figure: iteration 3

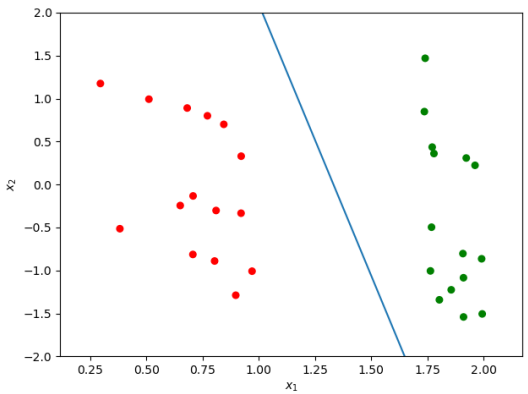


Figure: iteration 4

Basis Function

- Instead of working directly on the original input space, X, we build models in a new space transformed space $\Phi(X)$.
- $\Phi(X)$ is vector valued function, called basis function.
- For example, quadratic basis function for one dimensional input vector is given by

$$\Phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \phi_3(x) \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$$

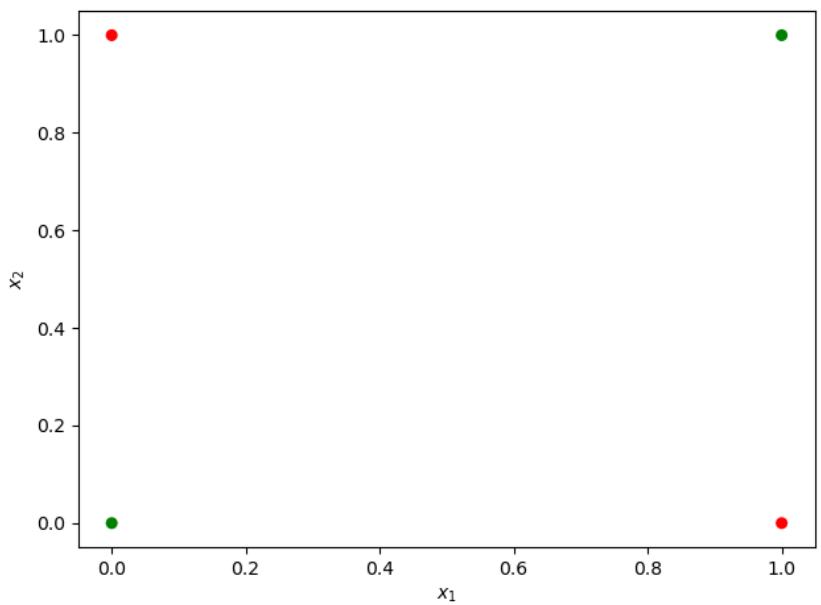
- Basis functions form the underlying support for our prediction function, for quadratic basis the prediction function is of the form

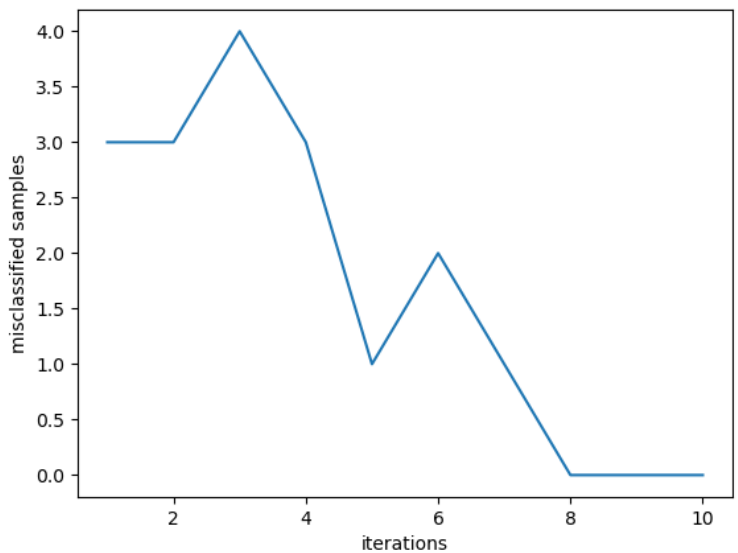
$$f(x) = w_0\phi_1(x) + w_1\phi_2(x) + w_2\phi_3(x),$$
$$\phi_1(x) = 1, \phi_2(x) = x, \phi_3(x) = x^2.$$

XOR example

The XOR function is 0 if both inputs are the same, or 1 if both inputs are different. XOR data is linearly not separable, but after using the following basis function, the Perceptron algorithm converges after 8 iterations.

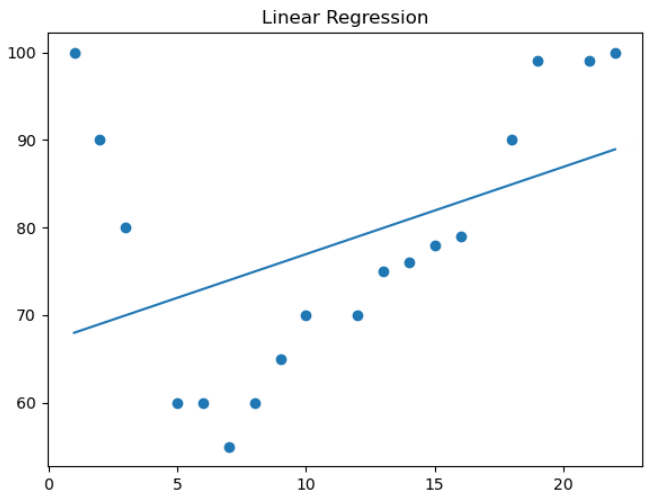
$$\phi_1(x) = 1, \phi_2(x) = x_1, \phi_3(x) = x_2, \phi_4(x) = x_1 * x_2,$$

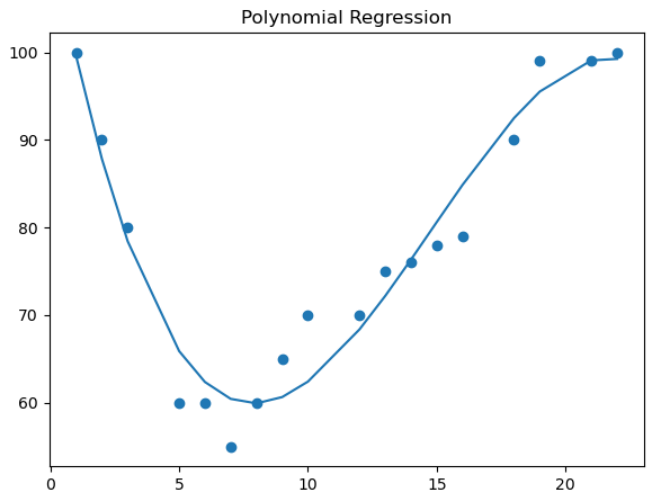




Polynomial Regression

$$f(x) = w_0\phi_1(x) + w_1\phi_2(x) + w_2\phi_3(x) + w_3\phi_4(x),$$
$$\phi_1(x) = 1, \phi_2(x) = x, \phi_3(x) = x^2, \phi_4(x) = x^3,$$





Radial Basis Functions

The most commonly used Radial Basis Function (RBF) is Gaussian RBF, its kernel with center x_c is given by

$$\phi(x, x_c) = \exp\left(\frac{||x - x_c||^2}{2 * \sigma^2}\right),$$

Where σ controls the smoothness of a Gaussian RBF. If there are n observations in the training set and that each observation can be used as a center of an RBF, then there are n RBFs. Let $\phi(x_i, x_j) = \phi_{ij}$ Then

$$\Phi(x) = \begin{bmatrix} \phi_{11} & \cdots & \phi_{1n} \\ & \cdots & \\ \phi_{n1} & \cdots & \phi_{nn} \end{bmatrix}$$

Activation Functions

The activation functions are used to transform the input between the required values, like $(0, 1)$ or $(-1, 1)$ etc. The Activation Functions can be basically divided into 2 types:

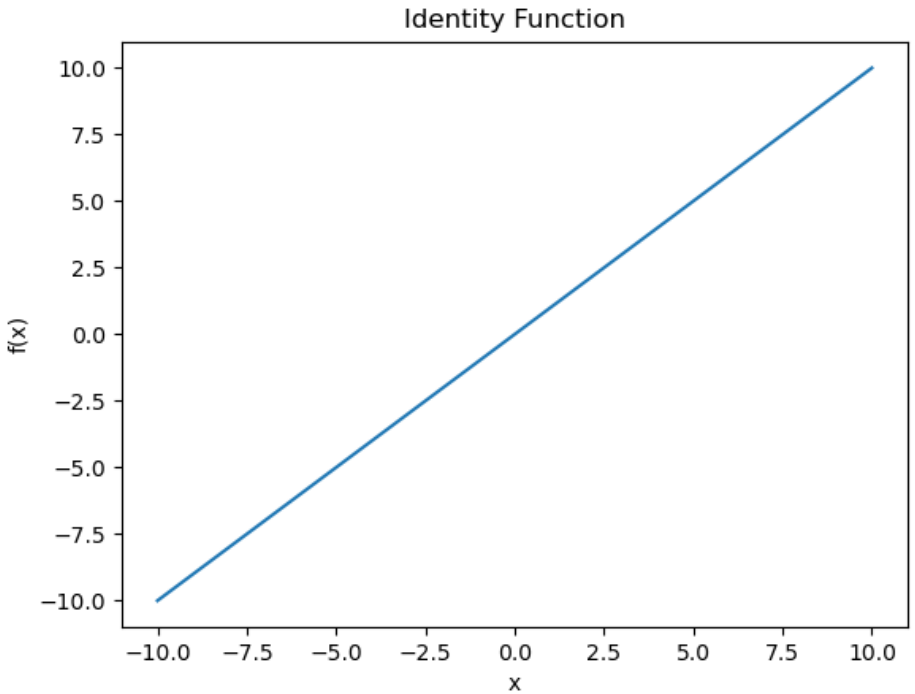
- Linear Activation Function
- Non-linear Activation Functions

Linear Activation Function

Function: $f(x) = x$

Range: $(-\infty, \infty)$

Derivative: 1

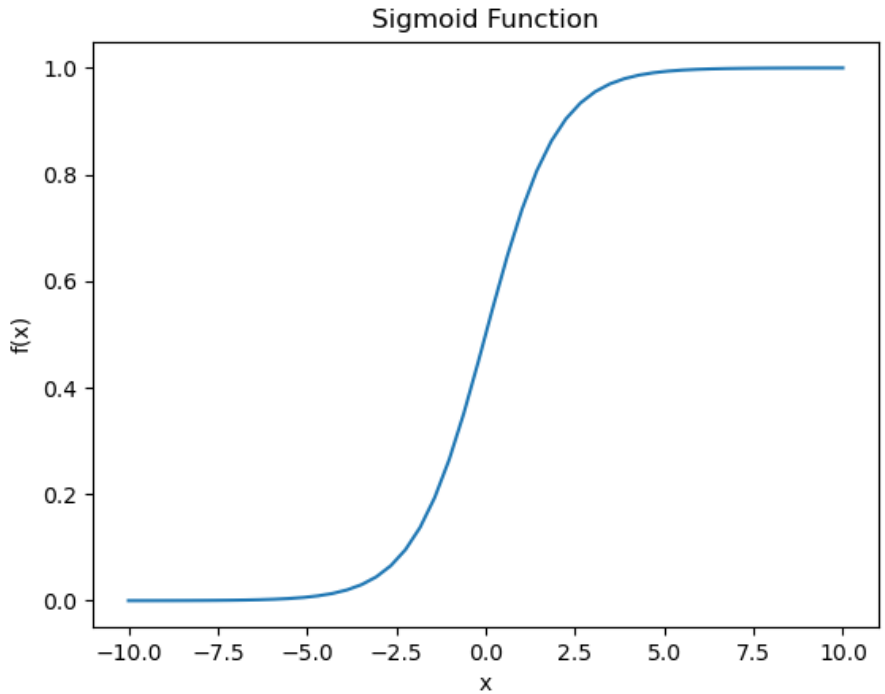


Nonlinear Activation Function: Sigmoid or Logistic Activation Function

Function: $f(x) = \dfrac{1}{1 + e^{-x}}$

Range: $(0, 1)$

Derivative: $f'(x) = f(x)(1 - f(x))$



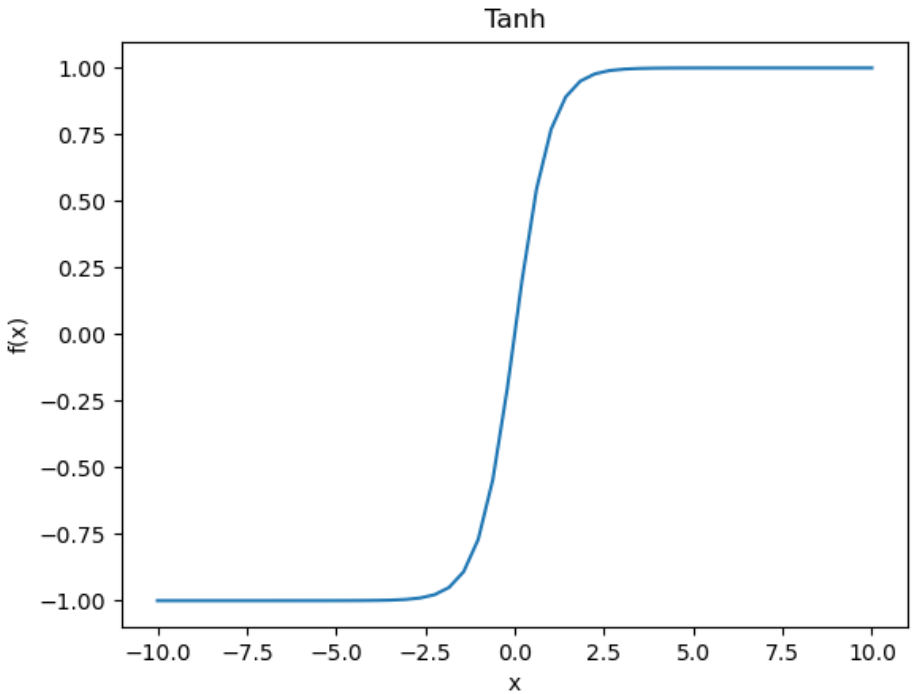
It is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1.

Tanh or hyperbolic tangent Activation Function

Function : $f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Range of : $(-1, 1)$

Derivative : $f'(x) = 1 - f(x)^2$

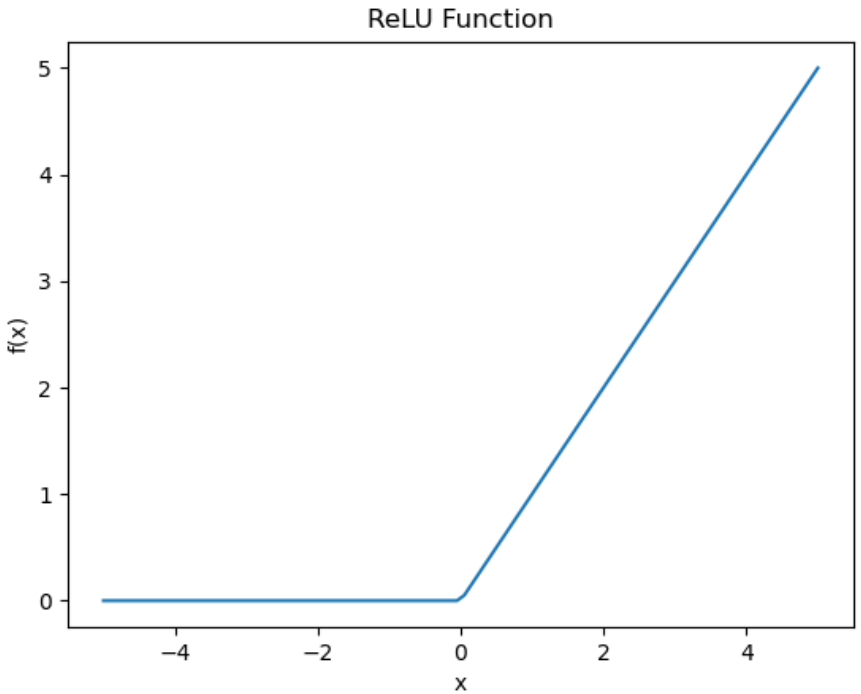


ReLU (Rectified Linear Unit) Activation Function

$$\text{Function}: f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$$

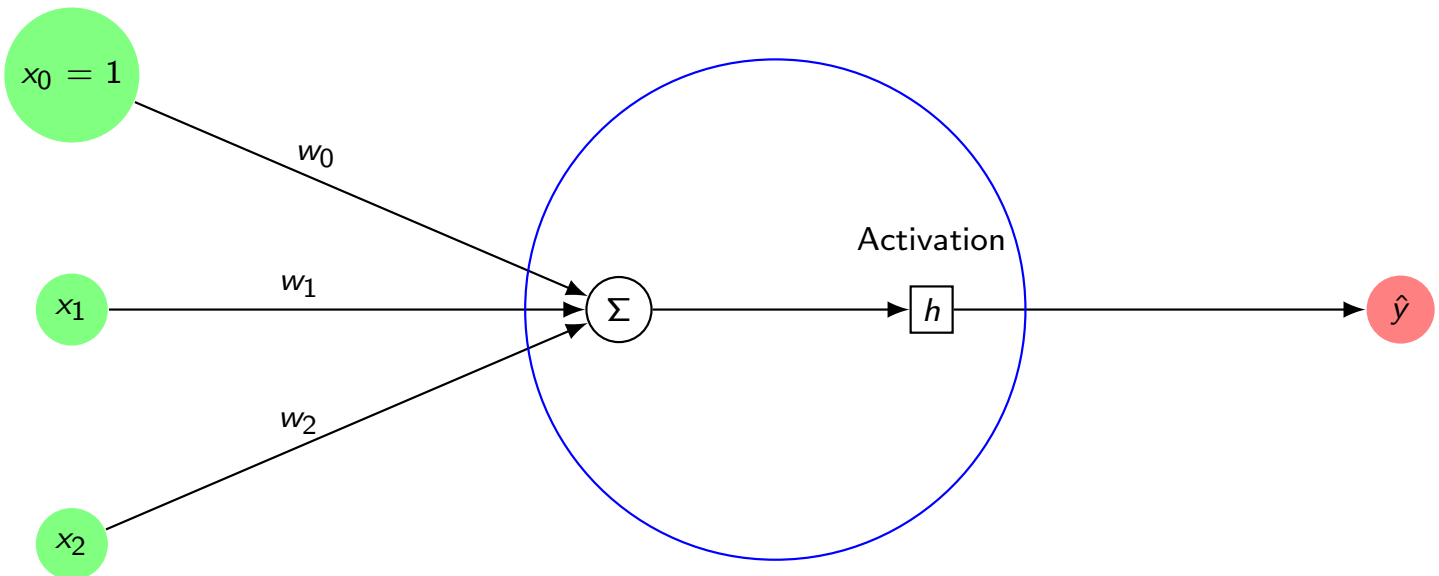
$$\text{Range}: (0, \infty)$$

$$\text{Derivative}: f'(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$$



Neurons in Neural Network

The basic building block of a neural network is a neuron. It is composed of four parts: inputs, weights and bias, total sum, and activation function.

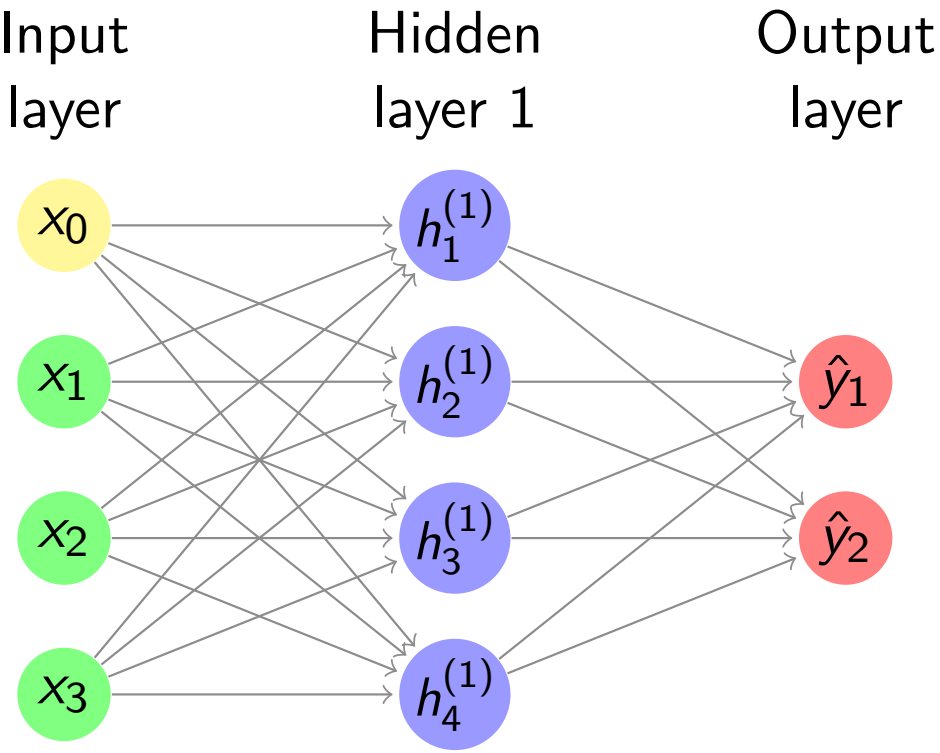


We can say that perceptron is a neural network with a single neuron.

Neural Networks

A typical neural network consists of layers of neurons. There are three types of layers:

- (One) input layer
- (One or more) hidden layer
- (One) output layer



Layers of Neural Network

- Input Layer: Each node in this layer represents an individual feature.
- Hidden Layer: They are called hidden layers simply because they are positioned between the input and output layers. We can have more neurons in the hidden layer than in the input layer, if we want the input layer to be represented in more dimensions.
- Output Layer: The number of nodes in the output layer depends on the number of possible outputs or prediction classes.

Each connection between two nodes has an associated weight and each weight represents the strength of the connection between the two nodes.

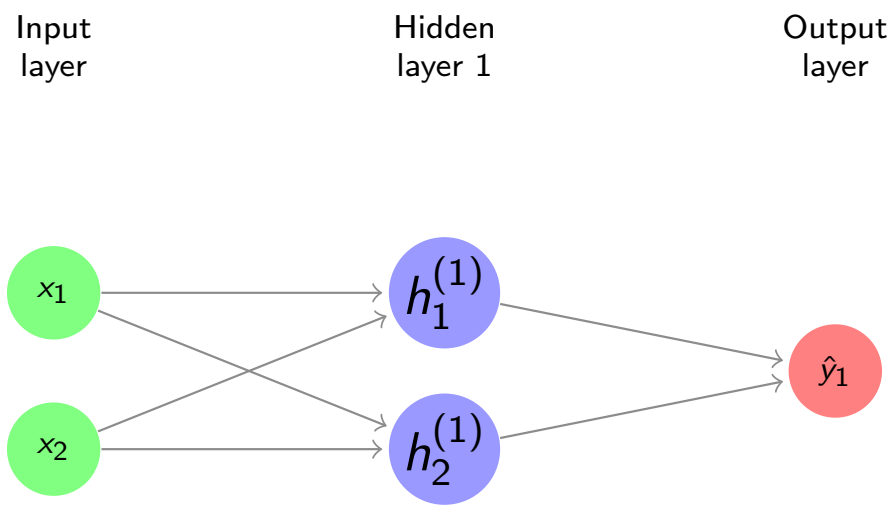
Feed-forward Neural Network

- Feed-forward Neural Network is also known as Muli Layer Perceptron (MLP) or Artificial Neural Networks
- Both forward and backward propagation are used to train a MLP.
- Forward Propagation: information flow starts from the input layer to the hidden layer(s) and finally to the output layer.
- Error Backward Propagation: information flow starts from the output layer to the hidden layer(s) and finally to the input layer.

Forward Propagation

For a regression problem with two features, consider a network with an input layer with two features, one hidden layer with two units and an output layer with one unit. We use *tanh* activation function and the squared error loss function. For simplicity we do not include the bias term.

$$RSS = L = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y})^2.$$



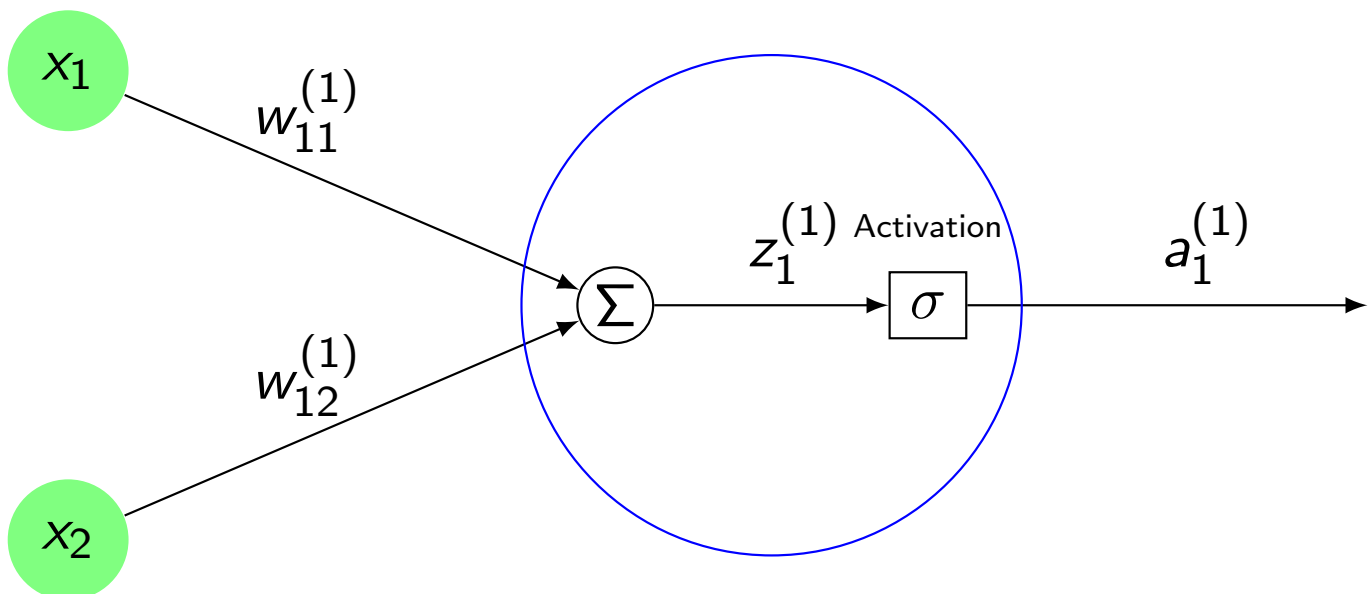
Forward Propagation

At each unit of hidden layer and output layer, the following operations are performed:

- Calculation of the weighted sum
- Application of activation function to the weighted sum

Forward Propagation: At unit 1 of hidden layer 1

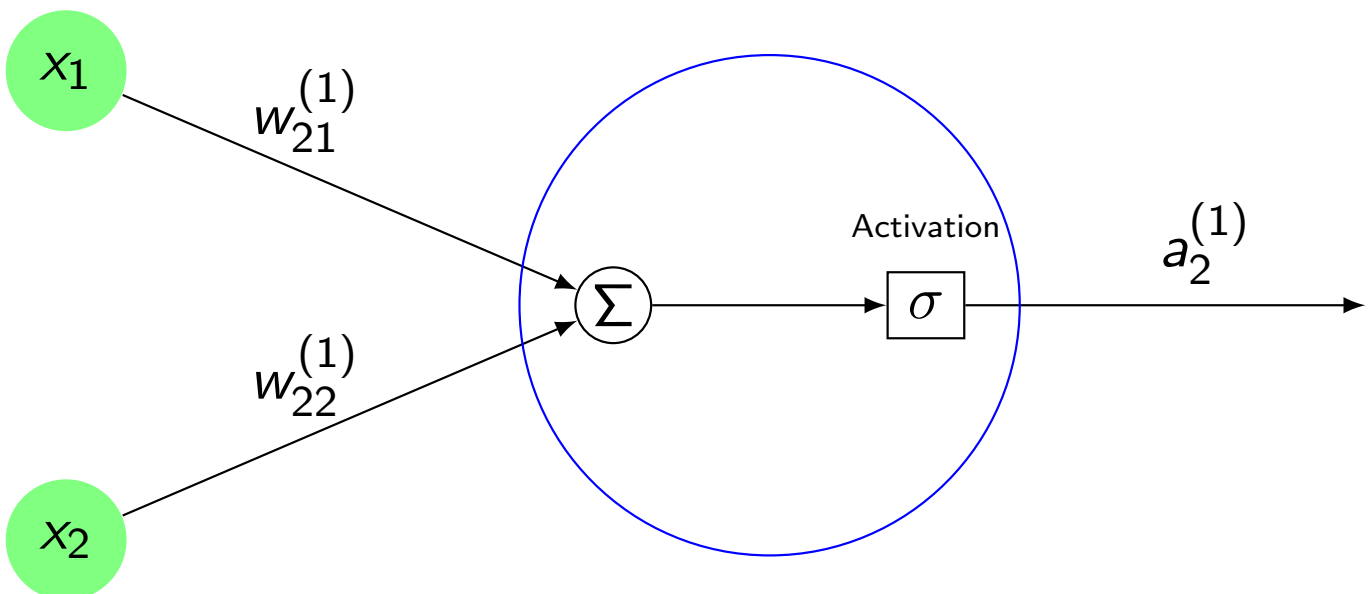
$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2$$
$$a_1^{(1)} = \tanh\left(z_1^{(1)}\right)$$



Superscript 1 corresponds to the hidden layer 1.

Forward Propagation: At unit 2 of hidden layer 1

$$z_2^{(1)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2$$
$$a_2^{(1)} = \tanh(z_2^{(1)})$$

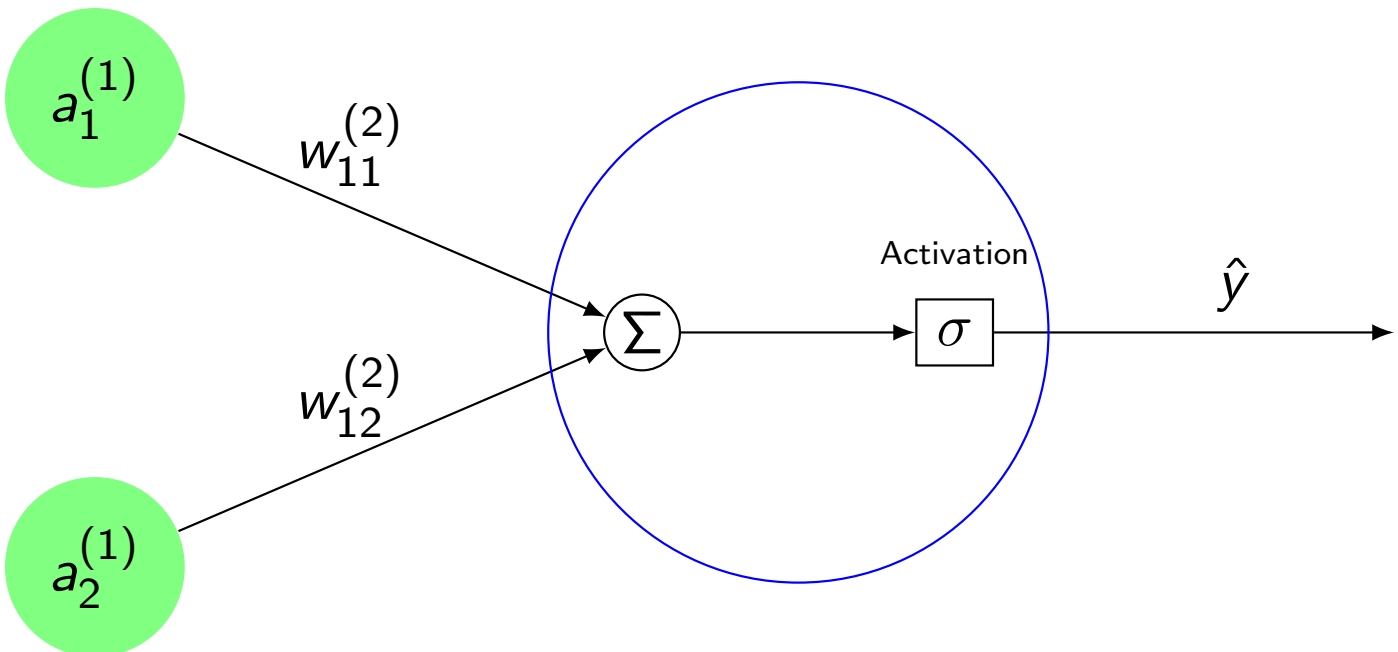


Superscript 1 corresponds to the hidden layer 1.

Forward Propagation: Output layer

$$z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)}$$

$$\hat{y} = a_1^{(2)} = \tanh(z_1^{(2)})$$



Superscript 2 corresponds to second layer of the network.

Forward Propagation: In Matrix Notations

$$\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \cdots \\ \boldsymbol{x}_N \end{bmatrix} \quad dim(X) = (N \times 2)$$

$$\boldsymbol{W}_1 = \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \end{bmatrix}$$

$$\boldsymbol{W}_2 = \begin{bmatrix} w_{11}^{(2)} \\ w_{12}^{(2)} \end{bmatrix}$$

$$\boldsymbol{Z}_1 = \boldsymbol{X} * \boldsymbol{W}_1, \quad dim(\boldsymbol{Z}_1) = (N \times 2) \times (2 \times 2) = (N \times 2)$$

$$\boldsymbol{A}_1 = tanh(\boldsymbol{Z}_1), \quad dim(\boldsymbol{A}_1) = dim(\boldsymbol{Z}_1) = (N \times 2)$$

$$\boldsymbol{Z}_2 = \boldsymbol{A}_1 * \boldsymbol{W}_2, \quad dim(\boldsymbol{Z}_2) = (N \times 2) \times (2 \times 1) = (N \times 1)$$

$$\boldsymbol{A}_2 = \boldsymbol{Z}_2, \quad dim(\boldsymbol{A}_2) = dim(\boldsymbol{Z}_2) = (N \times 1)$$

Error Back Propagation (Backpropagation)

- Error back propagation is used to improve prediction accuracy by adjusting the weights.
- We use gradient descent method to learn the weight of each node. The derivative of the error is computed, and the weight is adjusted by a negative multiple of this derivative.
- Starting at the output layer, the derivatives are calculated and back propagated to the input layer.

Backpropagation: Output Layer

We calculate the derivative of the loss function with respect to the weight $\boldsymbol{W}_2$.

$$L = \frac{1}{2}||(\boldsymbol{y} - \boldsymbol{A}_2)||_2^2$$

$$\frac{\partial L}{\partial \boldsymbol{W}_2} = \frac{\partial L}{\partial \boldsymbol{A}_2}\frac{\partial \boldsymbol{A}_2}{\partial \boldsymbol{Z}_2}\frac{\partial \boldsymbol{Z}_2}{\partial \boldsymbol{W}_2}$$

$$\frac{\partial L}{\partial \boldsymbol{A}_2} = (\boldsymbol{A}_2 - \boldsymbol{y})$$

$$\frac{\partial \boldsymbol{A}_2}{\partial \boldsymbol{Z}_2} = 1 \qquad (\boldsymbol{A}_2 = \boldsymbol{Z}_2)$$

$$\frac{\partial \boldsymbol{Z}_2}{\partial \boldsymbol{W}_2} = \boldsymbol{A}_1 \qquad (\boldsymbol{Z}_2 = \boldsymbol{A}_1 * \boldsymbol{W}_2)$$

$$\frac{\partial L}{\partial \boldsymbol{W}_2} = (\boldsymbol{A}_2 - \boldsymbol{y}) * 1 * \boldsymbol{A}_1$$

Backpropagation: Hidden Layer

We calculate the derivative of the loss function with respect to the weight matrix $\boldsymbol{W}_1$.

$$L = \frac{1}{2}||(\boldsymbol{y} - \boldsymbol{A}_2)||_2^2$$

$$\frac{\partial L}{\partial \boldsymbol{W}_1} = \frac{\partial L}{\partial \boldsymbol{A}_2}\frac{\partial \boldsymbol{A}_2}{\partial \boldsymbol{Z}_2}\frac{\partial \boldsymbol{Z}_2}{\partial \boldsymbol{A}_1}\frac{\partial \boldsymbol{A}_1}{\partial \boldsymbol{Z}_1}\frac{\partial \boldsymbol{Z}_1}{\partial \boldsymbol{W}_1}$$

$$\frac{\partial L}{\partial \boldsymbol{A}_2} = (\boldsymbol{A}_2 - \boldsymbol{y})$$

$$\frac{\partial \boldsymbol{A}_2}{\partial \boldsymbol{Z}_2} = 1 \qquad (\boldsymbol{A}_2 = \boldsymbol{Z}_2)$$

$$\frac{\partial \boldsymbol{A}_2}{\partial \boldsymbol{A}_1} = \boldsymbol{W}_2 \qquad (\boldsymbol{Z}_2 = \boldsymbol{A}_1 * \boldsymbol{W}_2)$$

$$\frac{\partial \boldsymbol{A}_1}{\partial \boldsymbol{Z}_1} = (1 - \boldsymbol{A}_1^2) \qquad (\boldsymbol{A}_1 = tanh(\boldsymbol{Z}_1))$$

$$\frac{\partial \boldsymbol{Z}_1}{\partial \boldsymbol{W}_1} = \boldsymbol{X} \qquad (\boldsymbol{Z}_1 = \boldsymbol{X} * \boldsymbol{W}_1)$$

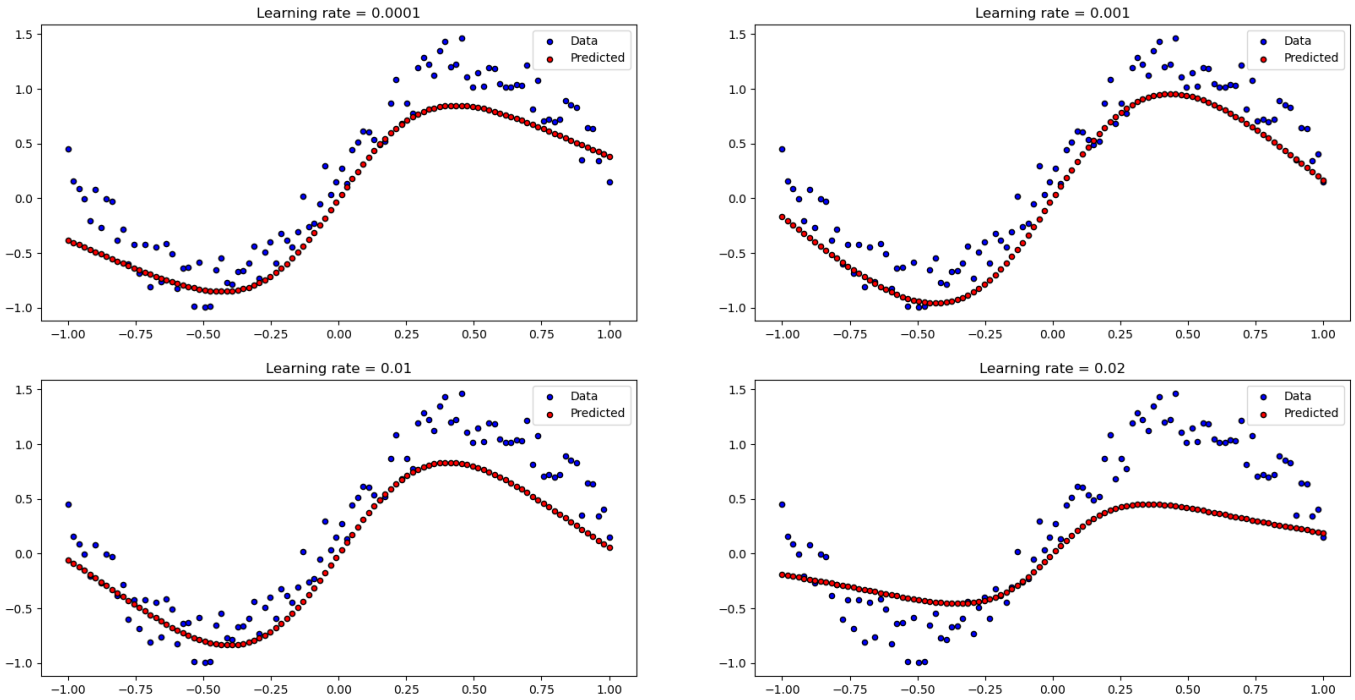
$$\frac{\partial L}{\partial \boldsymbol{W}_1} = (\boldsymbol{A}_2 - \boldsymbol{y}) * 1 * \boldsymbol{W}_2 * (1 - \boldsymbol{A}_1^2) * \boldsymbol{X}$$

Training an MLP

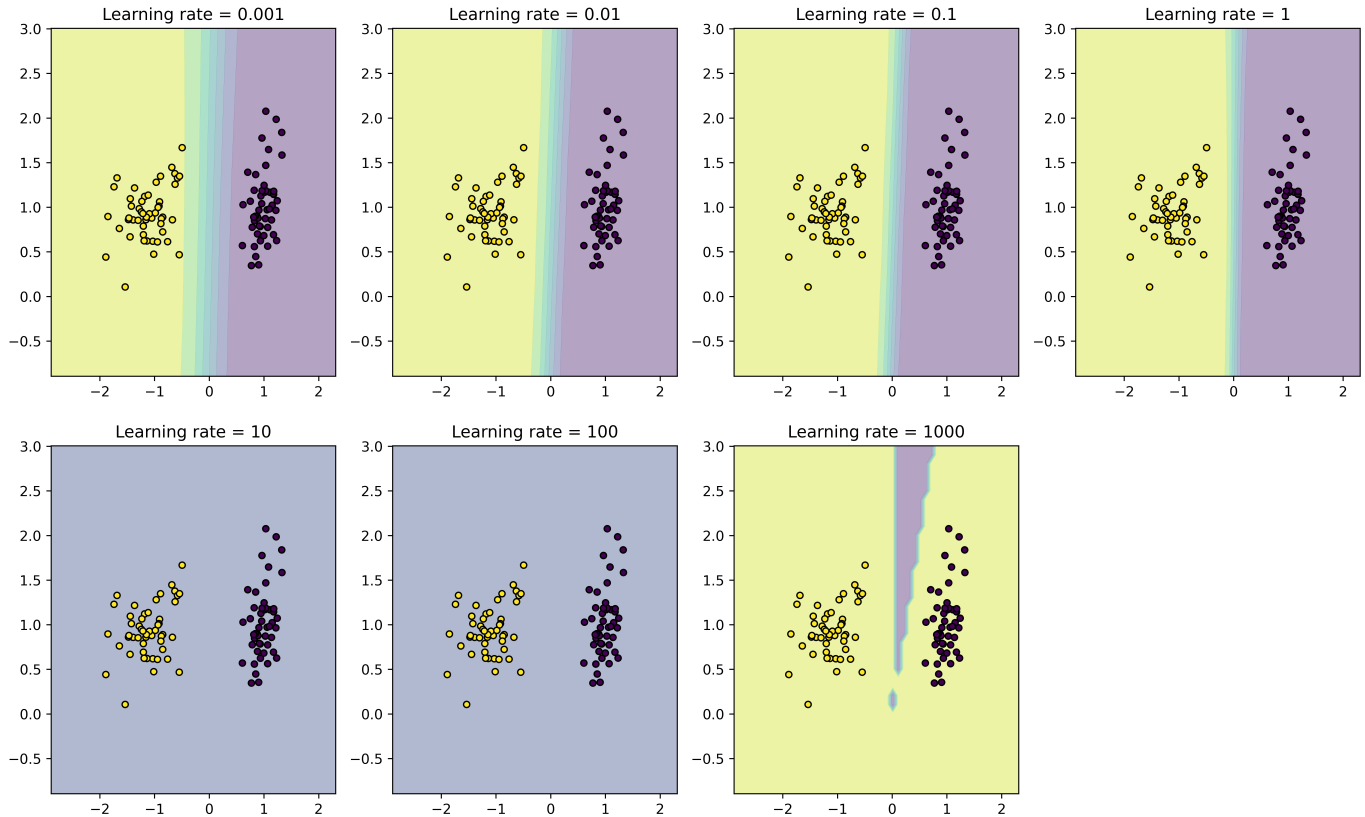
The following step are repeated for the given number of iterations.

- Begin from the input layer propagate data forward to the output layer.
- Calculate the error (the difference between the predicted and actual outcome).
- Back-propagate the error. Find derivative of the loss function with respect to each weight in the network, and update the model.

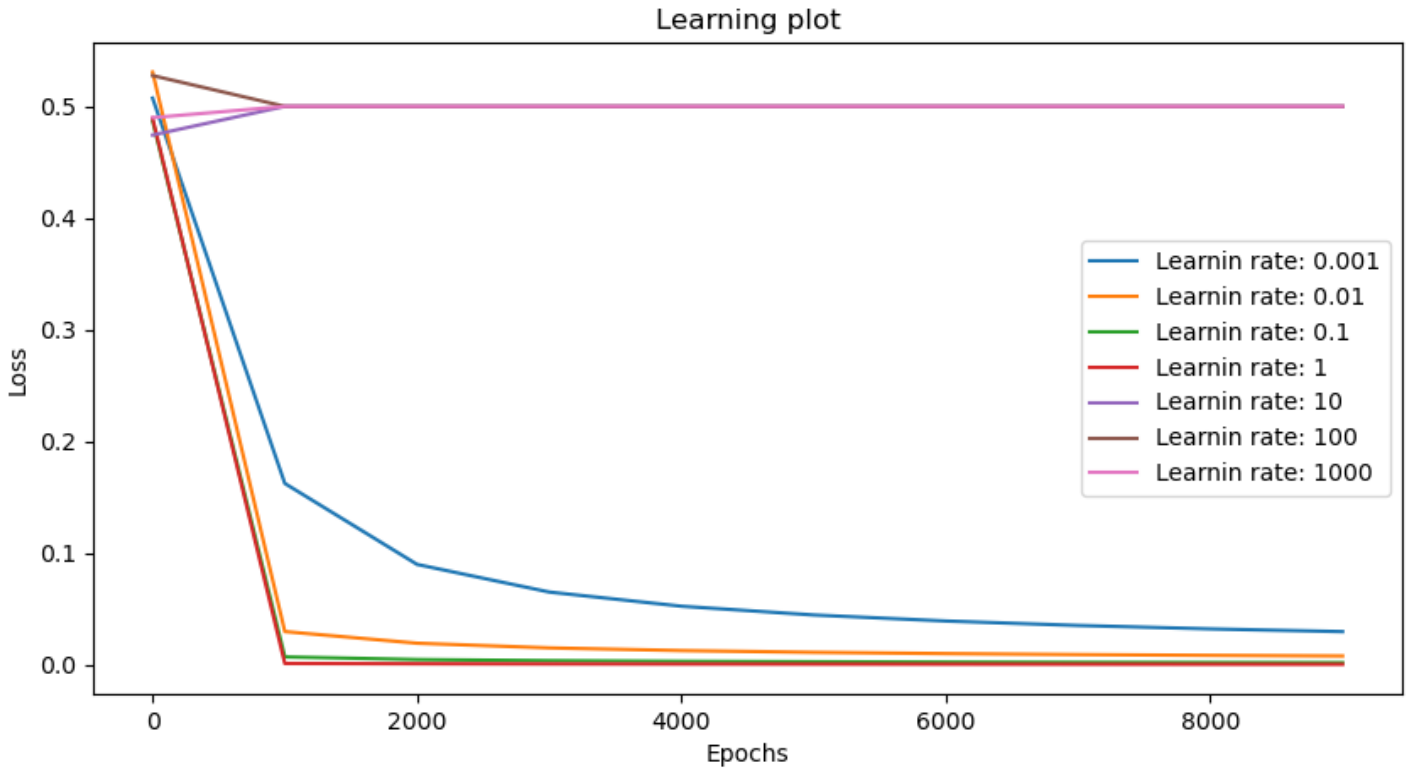
MLP: Regression Example



MLP: Classification Example

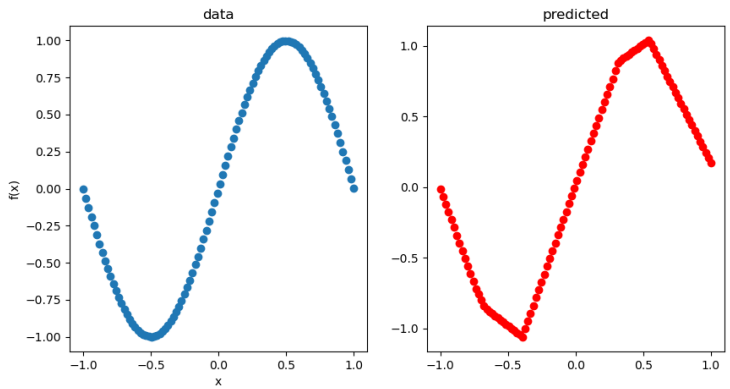


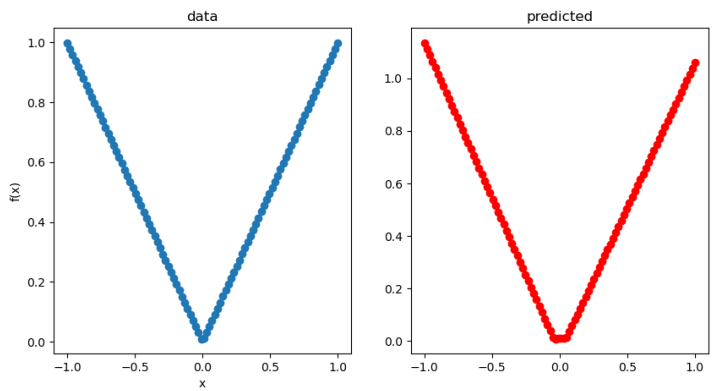
MLP: Classification Learning Plot

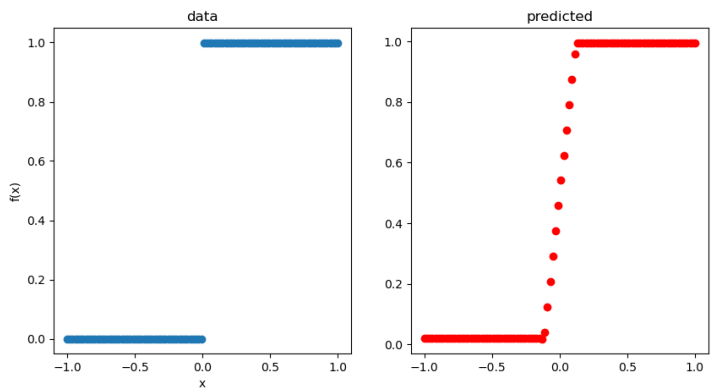


The Universal Approximation Theorem

The universal approximation theorem states that neural networks (with a single hidden layer) can approximate any continuous function.







Regularization

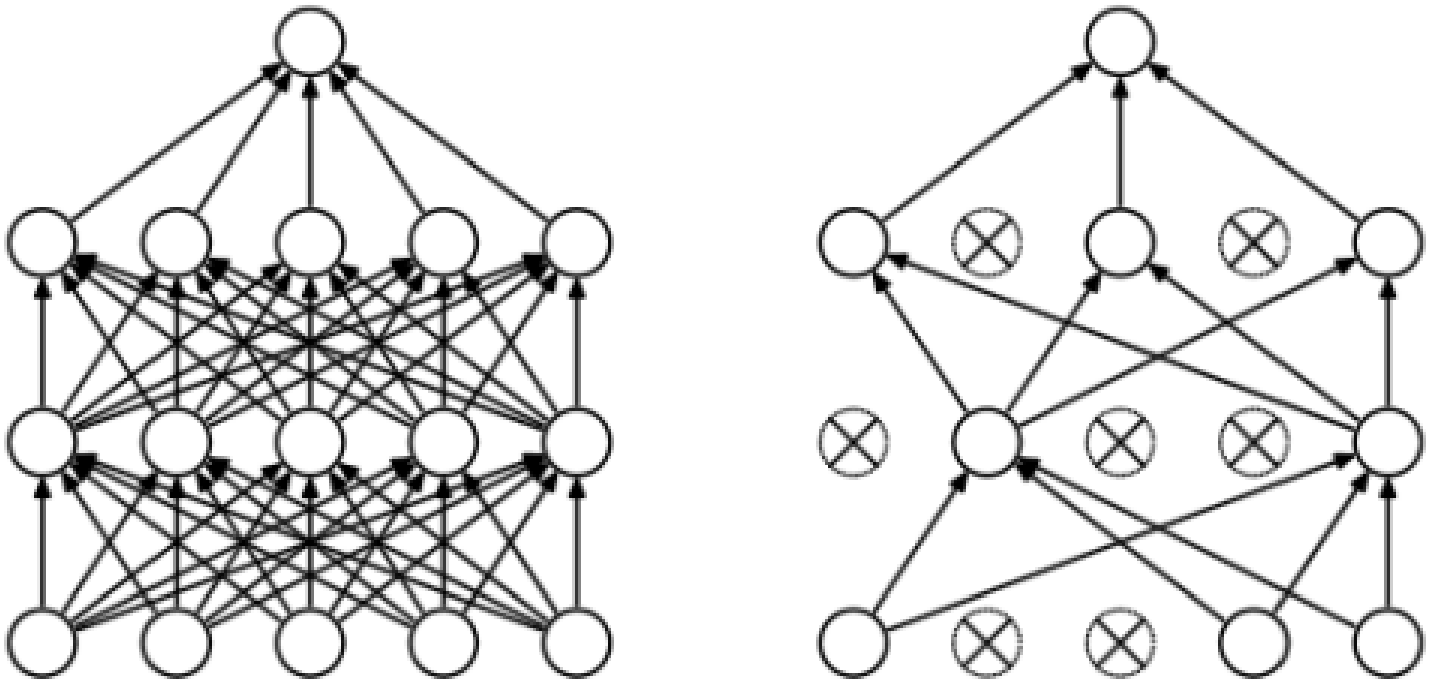
Some regularization techniques used for Multi-Layer Perceptrons:

- L_1 regularization
- L_2 regularization
- Dropout
- Early stopping
- Adversarial training

Regularization :Dropout

- Dropout is a common regularization technique used to prevent over-fitting in Neural Networks.
- Drop out corresponds to dropping out the nodes (from input and hidden layer).
- All the forward and backwards connections with a dropped node are temporarily removed.
- The nodes are dropped by a dropout probability of p.

Dropout



(a) Standard Neural Net

(b) After applying dropout.

The image is taken from Dropout paper: Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.

Summary

- Basis functions can help modeling non-linearity in the data while keeping linearity in parameters.
- The activation functions are used to transform the input between the required values, like (0, 1) or (-1, 1) etc.
- The basic building blocks of a neural network are neurons.
- A typical neural network consists of three layers of neurons: Input, Hidden and Output layers.
- Forward and backward propagation are iteratively performed to train a MLP.
- The universal approximation theorem states that neural networks (with a single hidden layer) can approximate any continuous function.
- Dropout is a common regularization technique used to prevent over-fitting in Neural Networks.