Module: **Machine Learning and Data-Driven Materials**

General Pipeline using Extreme Learning Machine – A Learning approach in neural networks

Week 5 – Session 2 (UK Time)

Institute for Materials Discovery (IMD)
University College London (UCL)
Dr. Adrian Rubio Solis, Prof. Adham Hashibon
Email: ucqsaru@ucl.ac.uk

Content: Week 7

Tuesday, Thursday :

Neural Networks Model:

- Neurons and the Brain: Model Representation
- General Pipeline using Extreme Learning Machine – A Learning approach in neural networks
- Regularisation

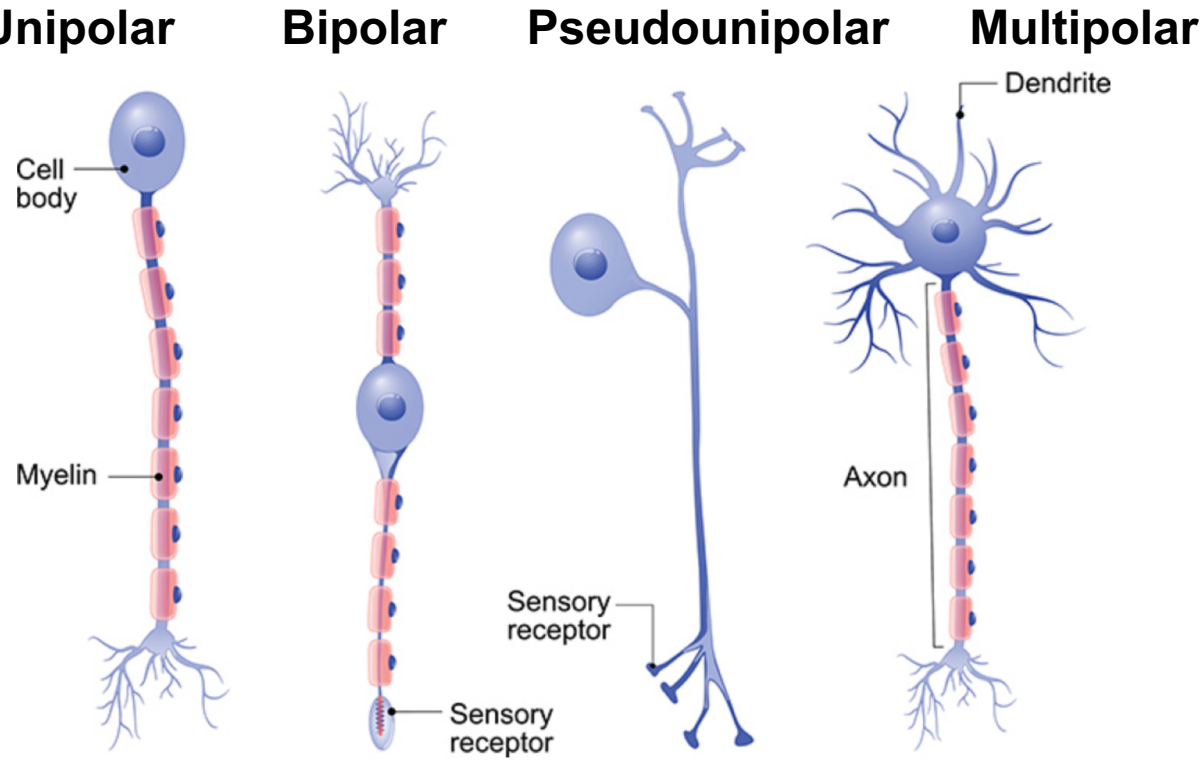
Gradient Descent Approach

- Perceptron Neural Network: A practical Case
- Example and intuitions: Batch vs on-line learning

Biological Neuron, Types and Mathematical Model

Neurons make up the nervous system and the brain. They are fundamental processing units that send and receive information that allows to move our muscles. For the spinal cord, then, we can say there are three types of neurons:

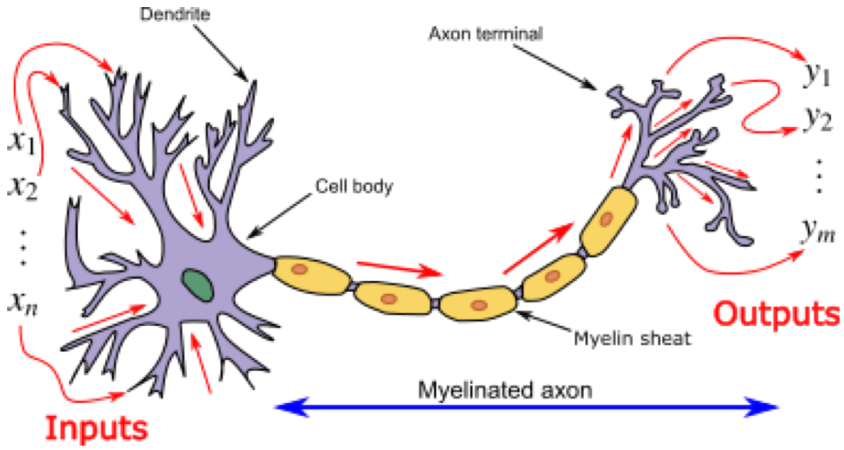
- Sensory Neurons
- Motor Neurons
- Interneurons



In the brain, the distinction between types neurons is much more complex. While in the spinal cord it is possible to distinguish three types of neurons, in the brain is not the case. Certainly there are brain neurons in sensory processing – like those in visual, auditory cortex and sensory processing. However, any of these **sensory or motor regions**, there hundreds of these types.

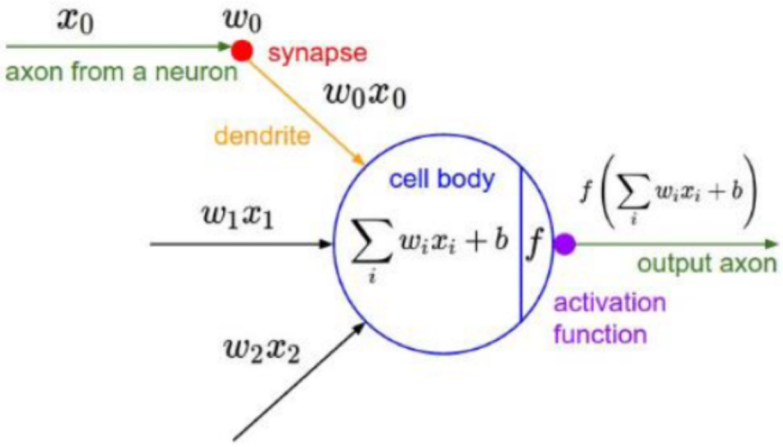
Furthermore, these different neurons have different electrical properties, different shapes, different genes expressed, different projection patterns and receive different inputs. In other words, a particular *combination* of features is one way of defining a neuron type.

Artificial Neurons, a model from nature

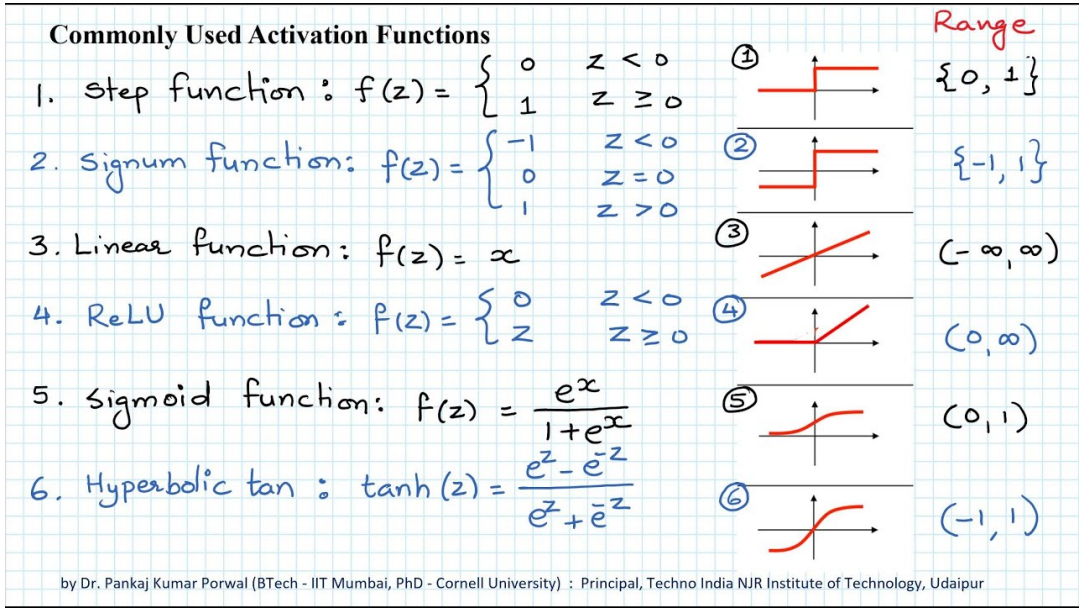


a) Biological Neuron

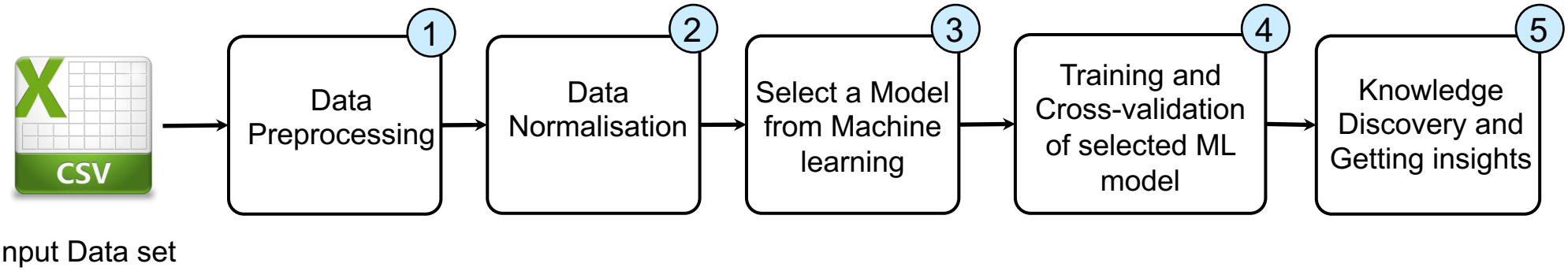
- Information usually comes from left to right as shown in figure
- Dendrites are the area where neurons receive most of all information, specially from other neurons whose signals come in the form of chemicals called neurotransmitters.
- Those signals picked up by the dendrites cause electrical changes in a neuron that are interpreted in an area called SOMA or CELL BODY
- The SOMA contains a nucleus which contains the DNA of the cell
- This information is then passed onto the AXON



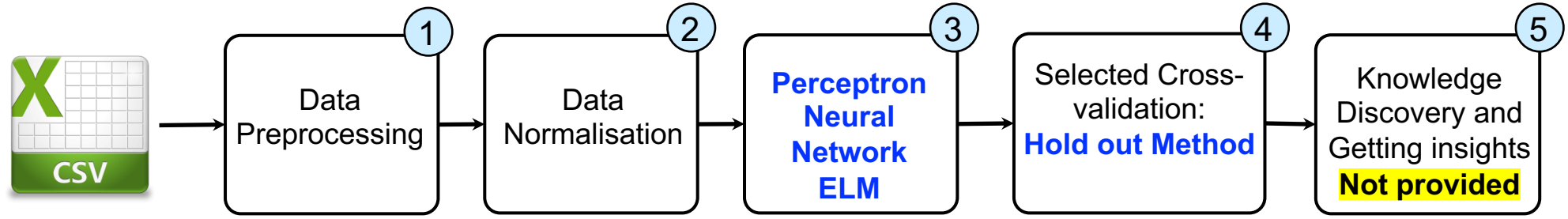
b) Artificial Neuron



General Pipeline in Machine Learning



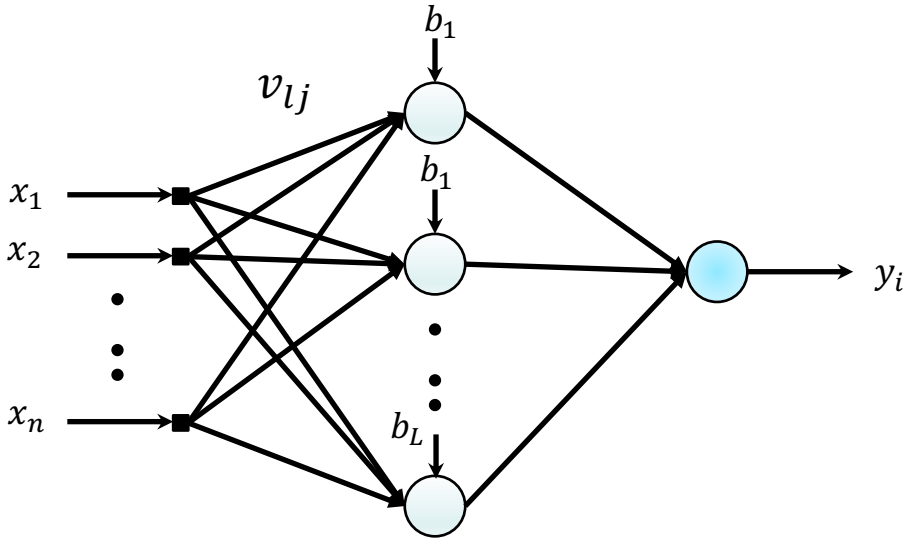
Input Data set



Input Data set

Notation used for Regression

- **Regression:** Root-Mean-Square-Error (RMSE)



$$RMSE = \sqrt{\frac{1}{P}\sum_{p=1}^{P}\left(y_p - \hat{y}_p\right)^2}$$

The output of the two-layer NN is given by the equation:

$$y_i = \sigma\left(\sum_{l=1}^{L} w_{il}\,\sigma\left(\sum_{j=1}^{n} v_{lj}x_j + v_{l0}\right) + w_{i0}\right); i = 1,2,\ldots m \qquad (2.1)$$

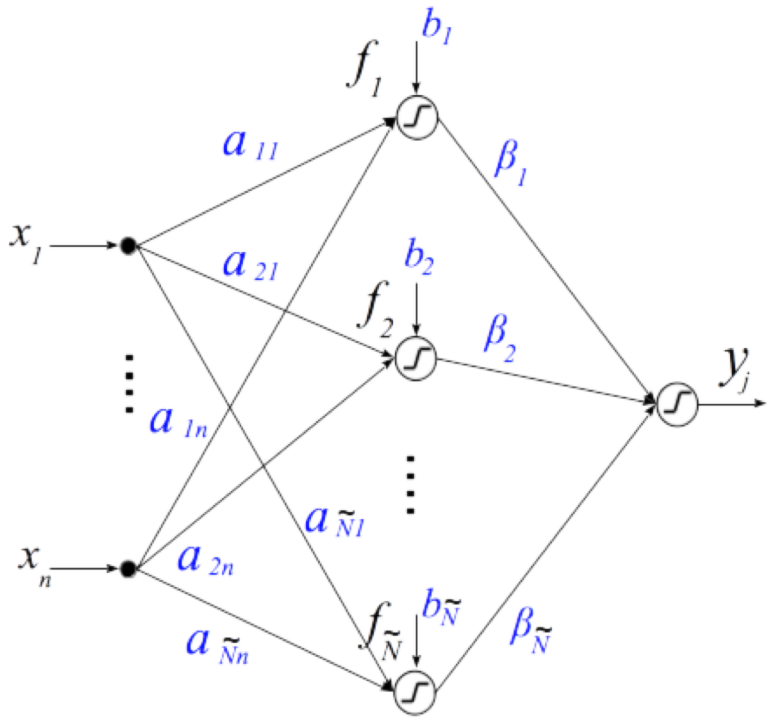
Defining the hidden-layer outputs z_l allows one to write:

$$z_l = \sigma\left(\sum_{j=1}^{n} v_{lj}x_j + v_{l0}\right); l = 1,2,\ldots L \qquad (2.2)$$

General Pipeline using Extreme Learning Machine – A Learning approach using Neural Networks

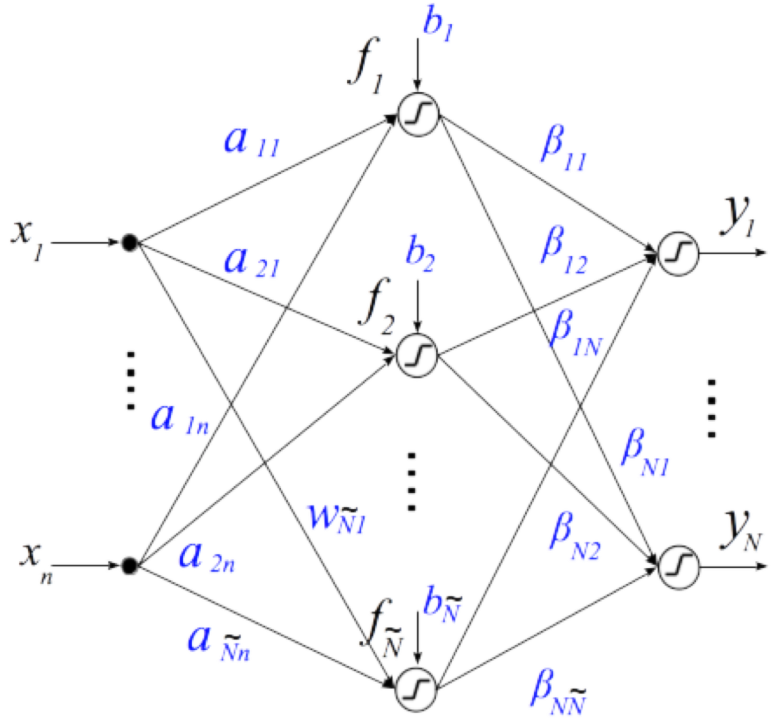
UCL INSTITUTE for MATERIALS DISCOVERY

Two Neural Structures! (both for supervised learning)



Multiple-Input-Single-Output

- To solve regression problems
- Faster learning
- One output
- Linear Model



Multiple-Input-Multiple-Output

- To solve Classification problems
- Faster
- One output
- Various linear Models

Training or Learning in Artificial Intelligence refers to the process applied to an artificial model to determine its optimal parameters to solve any of the four common problems in machine learning, i.e.

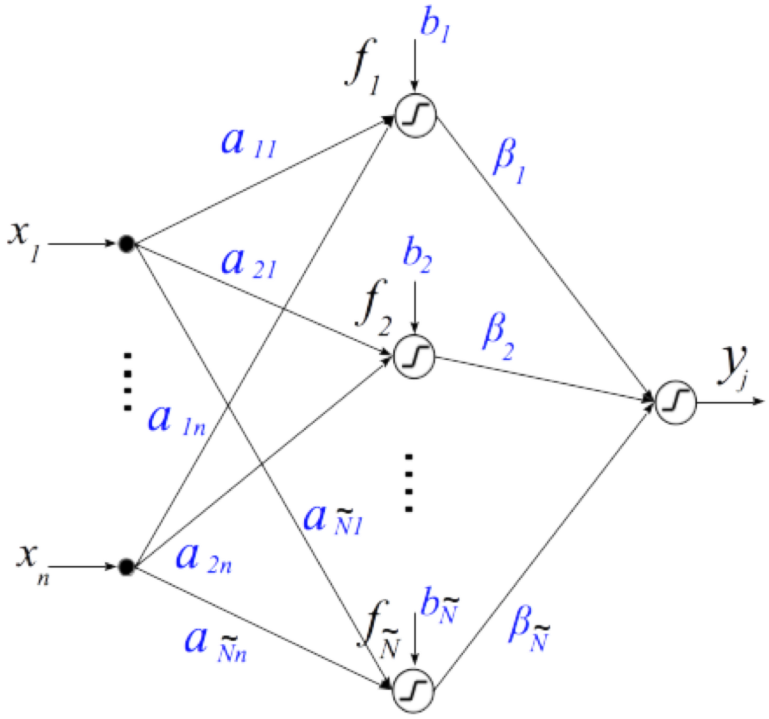
1) Regression
2) Classification
3) Data Clustering
4) Feature extraction

Cross-validation is the process used to validate how optimal (good) the parameters found in the process of learning/training are in the presence of new/unseen data (USUALLY called testing data).

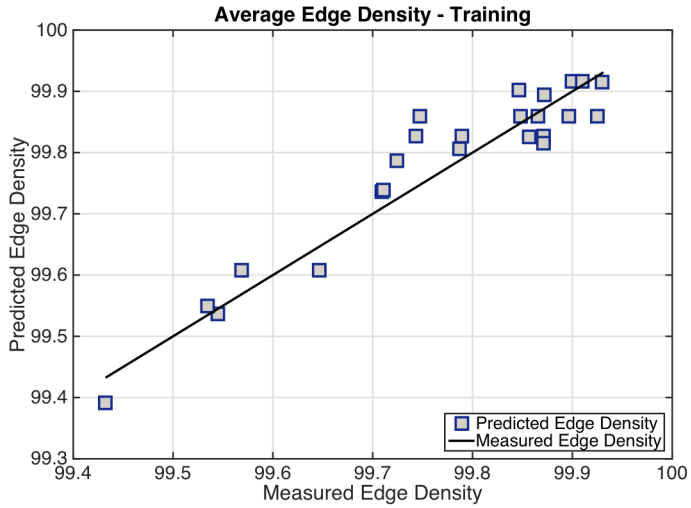
Extreme Learning Machine

UCL INSTITUTE for MATERIALS DISCOVERY

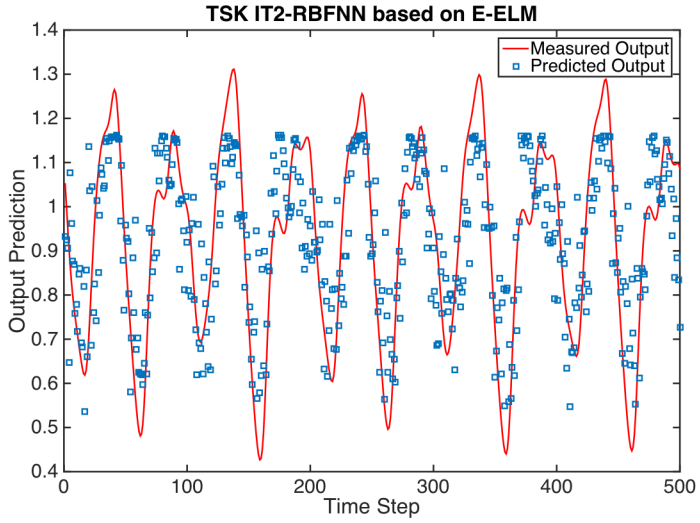
Regression



$$RMSE = \sqrt{\frac{1}{P}\sum_{p=1}^{P}(y_p - \hat{y}_p)^2}$$

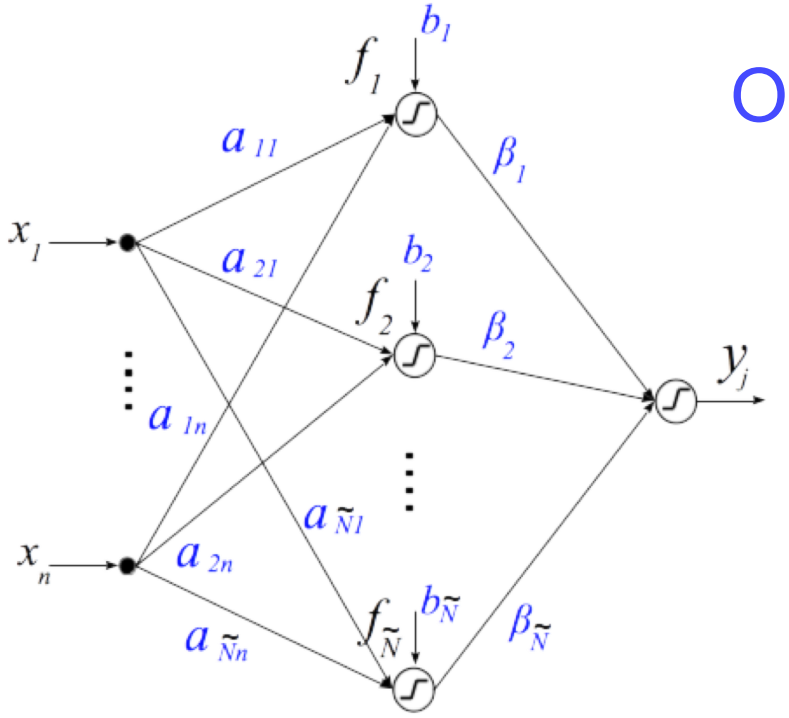


Data-Fit

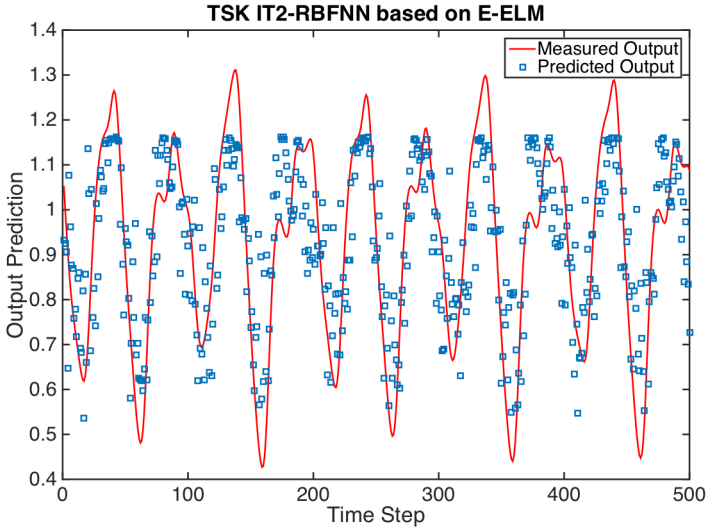


Time Series

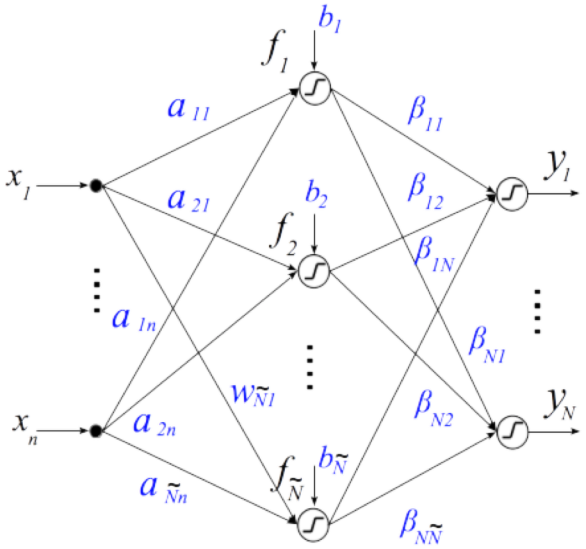
How data is fed into the Neural Structure?

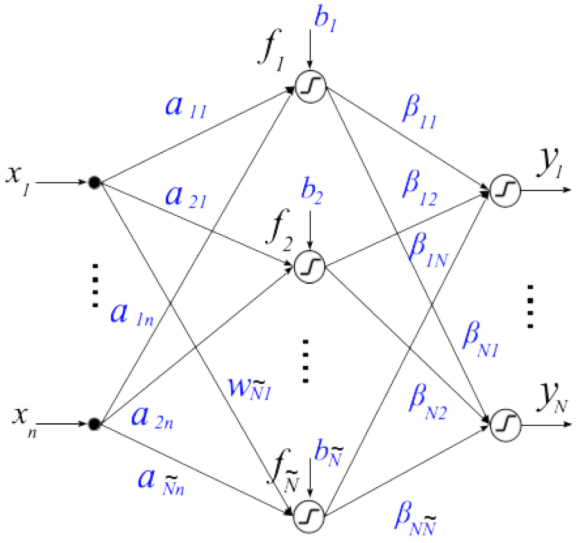


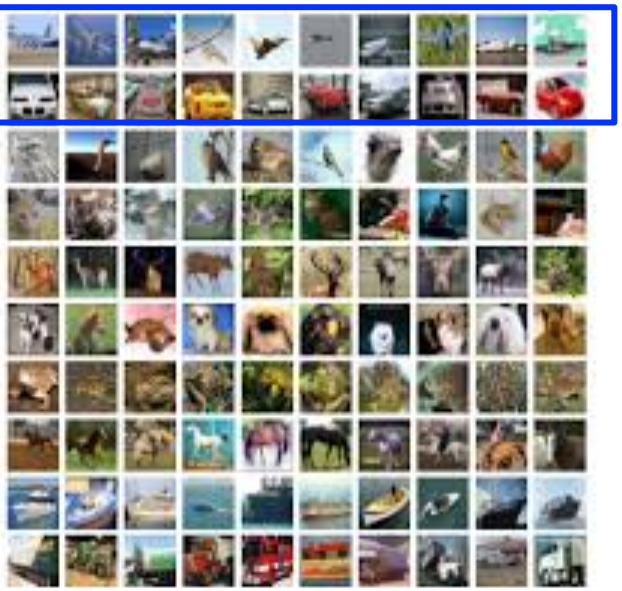
Output Target



Neural Configuration for Classification



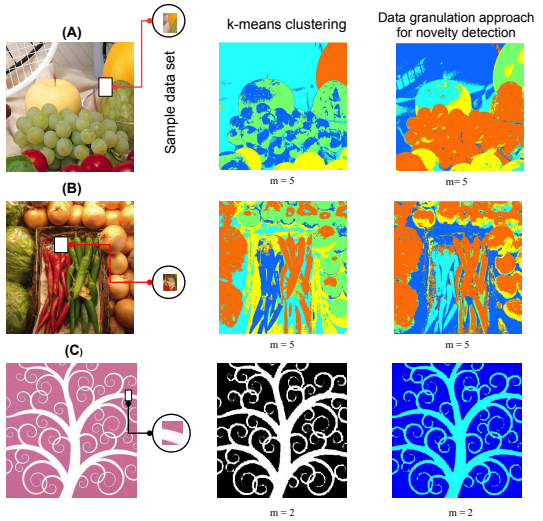


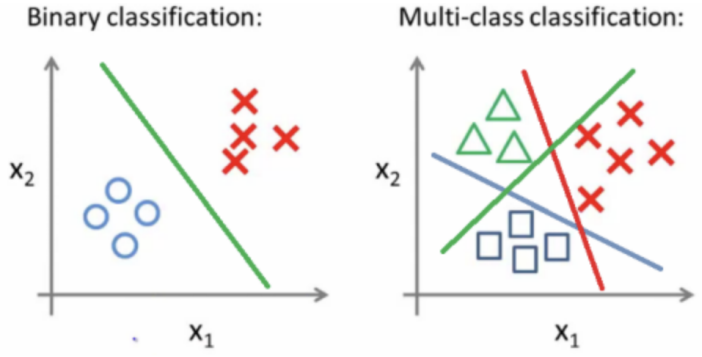


10 classes in total BINARY (Two classes) or MULTICLASS

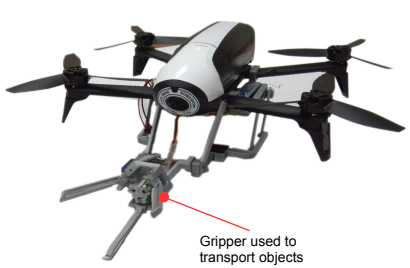
Airplane	Automobile	Truck
$y_1 = 1$	$y_1 = 0$	$y_1 = 0$
$y_2 = 0$	**$y_2 = 1$**	$y_2 = 0$
$y_3 = 0$	$y_3 = 0$	$y_3 = 0$
$y_4 = 0$	$y_4 = 0$	$y_4 = 0$
$y_5 = 0$	$y_5 = 0$	$y_5 = 0$
$y_6 = 0$	$y_6 = 0$	$y_6 = 0$
$y_7 = 0$	$y_7 = 0$	$y_7 = 0$
$y_8 = 0$	$y_8 = 0$	$y_8 = 0$
$y_9 = 0$	$y_9 = 0$	$y_9 = 0$
$y_{10} = 0$	$y_{10} = 0$	**$y_{10} = 1$**

Neural Configuration for Classification

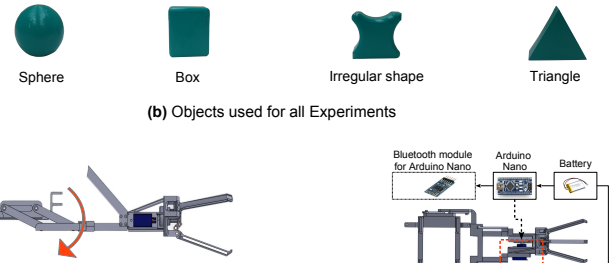




How to evaluate the accuracy of a model for Classification:



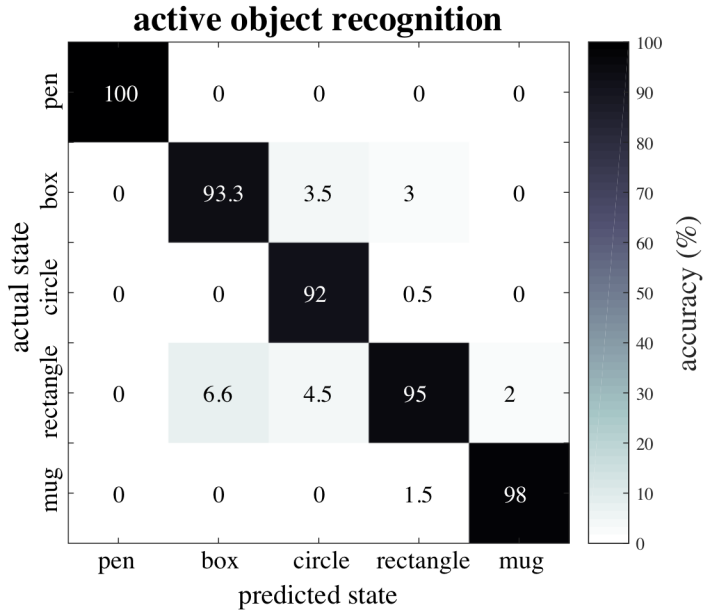
(a) Bebop Parrot 2: Unmanned Aerial Vehicle



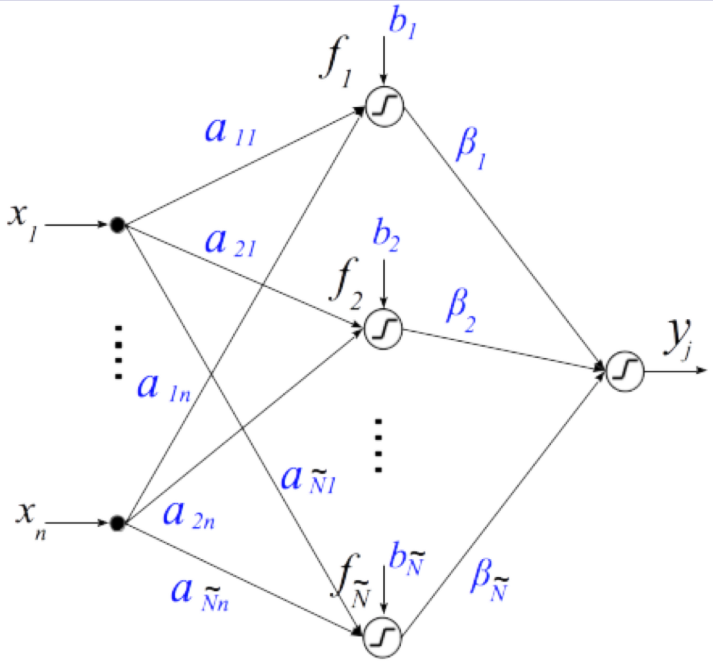
(b) Objects used for all Experiments

(c) Landing Extended Position Gripper

(d) Top View of Gripper



Model of an MISO NN (Perceptron)



Inputs **Outputs**

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2	Ref.No.	C	Cr	W	Si	V	Ta	Ti	N	B	TT	Tt(min)	YS	US	TE
3	1	0.12	8.73	2.09	0.25	0.25	0	0	0	0	750	60	549	659	12.3
4	1	0.1	8.72	2.09	0.23	0.23	0.07	0	0	0	750	60	544	652	12.3
5	1	0.092	8.32	0	0.15	0.2	0	0	0.054	0	760	60	539	630	13.3
6	2	0.1	9.3	0.93	0.11	0.22	0.094	0	0.002	0	650	120	674	783	13
7	2	0.1	9.3	0.93	0.11	0.22	0.094	0	0.002	0	750	120	538	657	15
8	2	0.1	9.3	0.95	0.13	0.23	0	0.056	0.002	0	650	120	573	698	16
9	2	0.1	9.3	0.95	0.13	0.23	0	0.056	0.002	0	750	120	464	605	23
10	3	0.1	9.3	2.22	0.12	0.052	0	0.0056	0.43	0	780	90	560	700	22
11	3	0.096	9.04	2.41	0.043	0.21	0	0.026	0.077	0	780	90	520	690	22
12	4	0.11	8.7	1	0	0.19	0.1	0	0	0	760	90	500	620	27
13	5	0.1	8.96	1.1	0.086	0.21	0.074	0	0	0	760	90	500	640	30
14	6	0.12	5.04	2.01	0.23	0.24	0	0	0.011	0	700	60	714	823	8.03
15	6	0.12	5.04	2.01	0.23	0.24	0	0	0.011	0	750	60	558	742	10
16	6	0.11	4.67	2.11	0.2	0.25	0.05	0	0.008	0	700	60	805	910	7.3
17	6	0.11	4.67	2.11	0.2	0.25	0.05	0	0.008	0	750	60	627	770	8.2
18	6	0.12	4.65	2.14	0.19	0.25	0.1	0	0.009	0	700	60	769	888	7.5
19	6	0.12	4.65	2.14	0.19	0.25	0.1	0	0.009	0	750	60	597	746	9.3
20	6	0.096	4.97	3	0.12	0.23	0	0	0.009	0	700	60	718	830	7.6
21	6	0.096	4.97	3	0.12	0.23	0	0	0.009	0	750	60	561	706	9.5
22	6	0.11	4.63	3.01	0.22	0.25	0.05	0	0.01	0	700	60	816	928	7.5
23	6	0.11	4.63	3.01	0.22	0.25	0.05	0	0.01	0	750	60	558	733	9.9
24	6	0.11	4.61	2.87	0.21	0.25	0.05	0	0.01	0.005	700	60	651	837	7.5
25	6	0.11	4.61	2.87	0.21	0.25	0.05	0	0.01	0.005	750	60	585	758	10.7
26	6	0.11	4.61	2.99	0.21	0.24	0.05	0	0.011	0.013	700	60	621	819	7.9
27	6	0.11	4.61	2.99	0.21	0.24	0.05	0	0.011	0.013	750	60	582	758	10.7

For N arbitrary distinct samples (x_i, t_i), where $x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T \in R^n$ and $t_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^T \in R^m$, what's more $(x_i, t_i) \in R^n \times R^m (i = 1, 2, \ldots, N)$, standard SLFNs with $\tilde{N}$ hidden nodes and activation function $f(x)$ are mathematically models as

$$\sum_{i=1}^{\tilde{N}} \beta_i f_i(x_j) = \sum_{i=1}^{\tilde{N}} \beta_i f(a_i \cdot x_j + b_i) = y_j \qquad j = 1, \ldots, N \quad (1)$$

Output Weight

Activation Function

Matrix representation of the MISO model

For N arbitrary distinct samples (x_i, t_i), where $x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T \in R^n$ and $t_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^T \in R^m$, what's more $(x_i, t_i) \in R^n \times R^m (i = 1, 2, \ldots, N)$, standard SLFNs with $\tilde{N}$ hidden nodes and activation function $f(x)$ are mathematically models as

$$\sum_{i=1}^{\tilde{N}} \beta_i f_i(x_j) = \sum_{i=1}^{\tilde{N}} \beta_i f(a_i \cdot x_j + b_i) = y_j \qquad j = 1, \ldots, N \quad (1)$$

where $a_i = [a_{i1}, a_{i2}, \ldots, a_{in}]^T$ is the weight vector connecting the i th hidden node and the input nodes, and b_i is the threshold of the ith hidden node.$\beta_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ is the weight vector connecting the ith hidden node and the output nodes.$a_i \cdot x_j$ represents the inner product of a_i and x_i, and the activation function usually choose "Sigmoid", "Sine", "RBF".

The above Eq. (1) can be written compactly as

$$H\beta = T \quad (2)$$

where $H(a_1, \ldots, a_{\tilde{N}}, b_1, \ldots, b_{\tilde{N}}, x_1, \ldots x_N)$

UCL INSTITUTE for MATERIALS DISCOVERY

Moore-Penrose Matrix

We have data $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_n)$ that we wish to fit with a linear combination of the columns of $\mathbf{X}$, where

$$\mathbf{X} = [\, \mathbf{1}\ \mathbf{X}_1\ \cdots\ \mathbf{X}_D\ ]$$

and $\mathbf{X}_d = \{X_{di}\}_{i=1}^n$ are the data on the dth variable, $d = 1, \ldots, D$. To find the "least squares solution" we need to find length vector $\mathbf{w}$ of length $(D+1)$ that minimizes the sum of squared errors $(\mathbf{Xw} - \mathbf{Y})'(\mathbf{Xw} - \mathbf{Y})$

We take the derivative of the sum of squared errors w.r.t. $\mathbf{w}$, set to zero and solve for $\hat{\mathbf{w}}$:

$$\begin{aligned}
\mathbf{0} &= \frac{\partial}{\partial \mathbf{w}}(\mathbf{Xw} - \mathbf{Y})'(\mathbf{Xw} - \mathbf{Y}) \\
&= \frac{\partial}{\partial \mathbf{w}}\left[\mathbf{w}'\mathbf{X}'\mathbf{Xw} - \mathbf{w}'\mathbf{X}'\mathbf{Y} - \mathbf{Y}'\mathbf{Xw} - \mathbf{Y}'\mathbf{Y}\right] \\
&= \frac{\partial}{\partial \mathbf{w}}\mathbf{w}'\mathbf{X}'\mathbf{Xw} - 2\frac{\partial}{\partial \mathbf{w}}\mathbf{w}'\mathbf{X}'\mathbf{Y} - 0 \\
&= 2\mathbf{X}'\mathbf{Xw} - 2\mathbf{X}'\mathbf{Y}.
\end{aligned}$$

This leads to the definition of the Least Squares "Normal Equations"

$$\mathbf{X}'\mathbf{Y} = \mathbf{X}'\mathbf{Xw}.$$

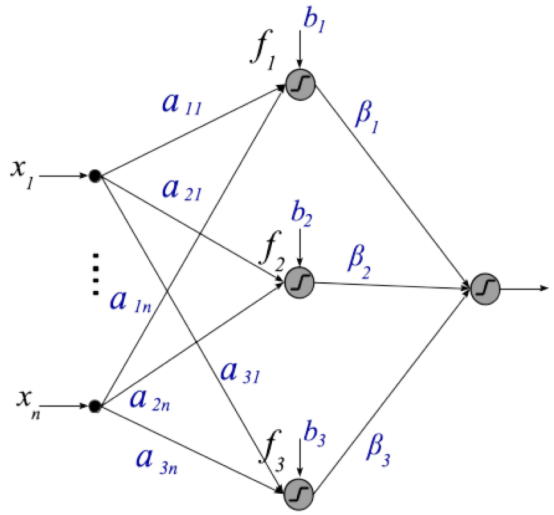
This system of $D + 1$ equations defines the Least Squares solutions... any vector $\mathbf{w}$ that satisfies it is a "Least Squares" estimate. Of course, if $\mathbf{X}$ is full rank, then we have the standard answer

$$\hat{\mathbf{w}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}.$$

Notes.

where $H(a_1, \ldots, a_{\tilde{N}}, b_1, \ldots, b_{\tilde{N}}, x_1, \ldots x_N)$

$$H = \begin{bmatrix} f(a_1 \cdot x_1 + b_1) & \cdots & f(a_{\tilde{N}} \cdot x_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ f(a_1 \cdot x_N + b_1) & \cdots & f(a_{\tilde{N}} \cdot x_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}}, \beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N} \times m}, T = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}_{N \times m}$$



Therefore, training SFLN is converted into solving linear equations $H\beta = T$ least squares solution.

$$\left\| H(a_1, \ldots, a_{\tilde{N}}, b_1, \ldots, b_{\tilde{N}})\hat{\beta} - T \right\| = \min_{\beta} \left\| H(a_1, \ldots, a_{\tilde{N}}, b_1, \ldots, b_{\tilde{N}})\beta - T \right\| \quad (3)$$

The above Eq. (3) least squares solution of the above liner system is

$$\hat{\beta} = H^{\dagger}T \quad (4)$$

Pseudoinverse matrix:
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4938676

Then Eq. (1) can be expressed as:

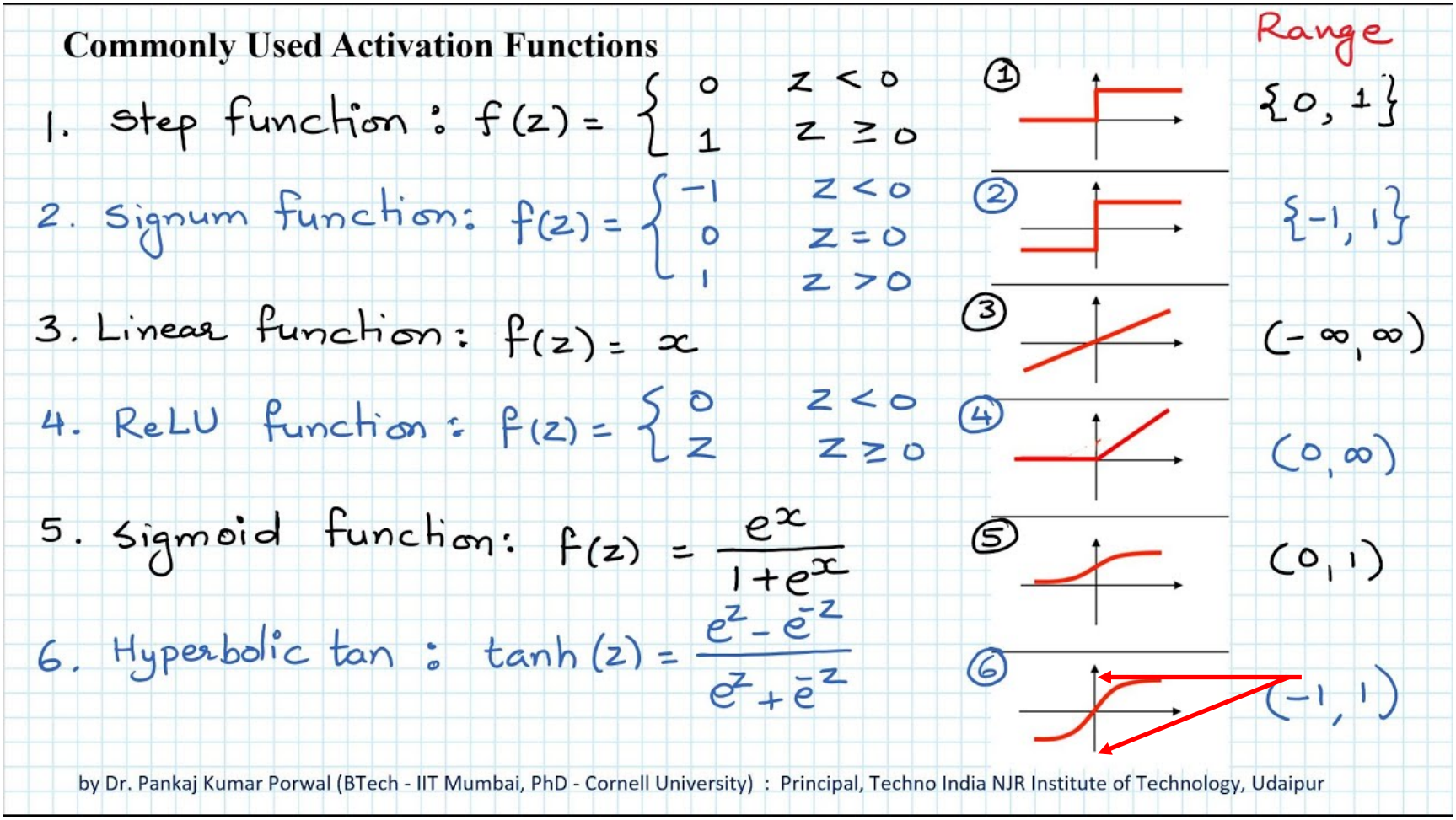
$$\sum_{i=1}^{\tilde{N}} \beta_i f_i(x_j) = \sum_{i=1}^{\tilde{N}} \beta_i f(a_i \cdot x_j + b_i) = t_j, \; j = 1, \ldots, N \quad (1)$$

Regularisation

Regularization is **a technique used for tuning the function by adding an additional penalty term in the error function**. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values.

In ELM theory, by adding a penalty constant and solving for the Lagragian Multipliers **[3]**:

$$\beta = \left(\frac{I}{\gamma} + \mathbf{H}^T \mathbf{H} \right)^{\dagger} \mathbf{H}^T \mathbf{T} \qquad (5)$$



Activation Functions used in the Perceptron NN

Extreme Learning Machine Algorithm

Thus, a simple learning method for SLFNs called extreme learning machine (ELM) can be summarized as follows:

Algorithm ELM: Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, and hidden node number $\tilde{N}$,

Step 1: Randomly assign input weight $\mathbf{w}_i$ and bias b_i, $i = 1, \ldots, \tilde{N}$.

Step 2: Calculate the hidden layer output matrix $\mathbf{H}$.

Step 3: Calculate the output weight β

$$\beta = \mathbf{H}^{\dagger}\mathbf{T}, \tag{16}$$

where $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^{\mathrm{T}}$.

$$\beta = \mathbf{H}^{\dagger}T = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T T$$

Example 1: $A^+ = A^T(AA^T)^{-1}$ if A is onto, i.e., has linearly independent rows (A is right invertible)

Example 2: $A^+ = (A^TA)^{-1}A^T$ if A is 1-1, i.e., has linearly independent columns (A is left invertible)

References:

[1] Huang G B, Zhu Q Y, Siew C K. Extreme learning machine: theory and applications[J]. Neurocomputing, 2006, 70(1-3): 489-501.

[2] Huang G B, Zhou H, Ding X, et al. Extreme learning machine for regression and multiclass classification[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2012, 42(2): 513-529.

[3] Deng, Wanyu, Qinghua Zheng, and Lin Chen. "Regularized extreme learning machine." *2009 IEEE symposium on computational intelligence and data mining*. IEEE, 2009.


Thank you