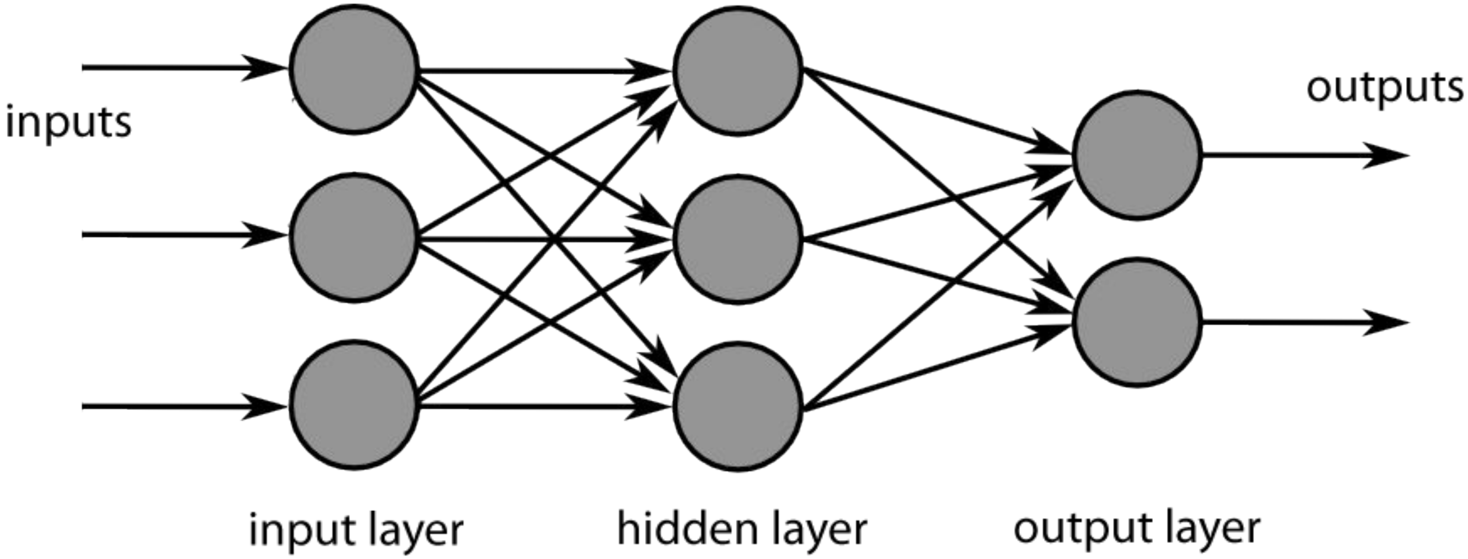
ANN and Backpropagation

Md. Samiullah, PhD
Assistant Professor,
Department of Computer Science & Engineering,
University of Dhaka

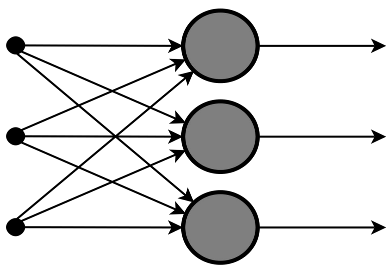
Introduction

- Robust approach to predict real, discrete or vector value
- Inspired from Biological learning system
- Solve both biological and non-biological problems
- Representation Element
  - Nodes
  - Edges
  - Edge Weights
  - Bias
- Instances are represented by many attribute-value pairs
- The training examples may contain errors
- Long training times are acceptable
- Fast evaluation of the learned target function may be required
- The ability of humans to understand the learned target function is not important

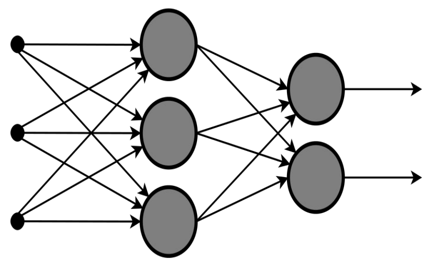
Introduction



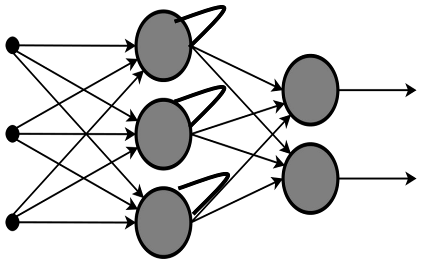
Types of Neural Networks



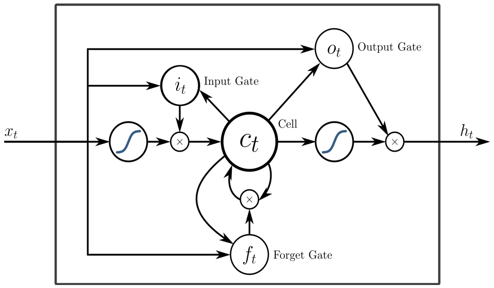
Perceptron or Single Layer NN



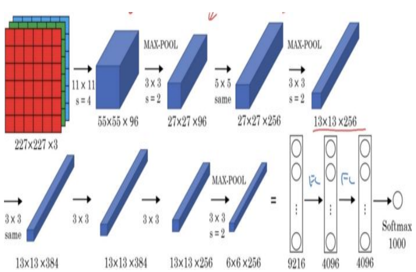
Feed Forward (MLP) NN



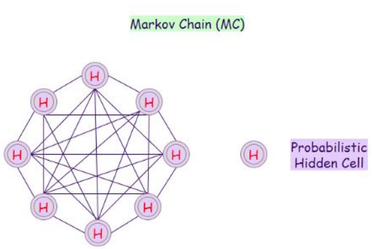
Recurrent (RNN) NN



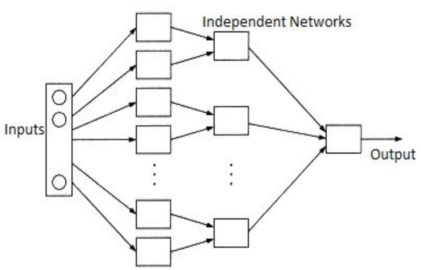
Long Short Term Memory (LSTM) NN



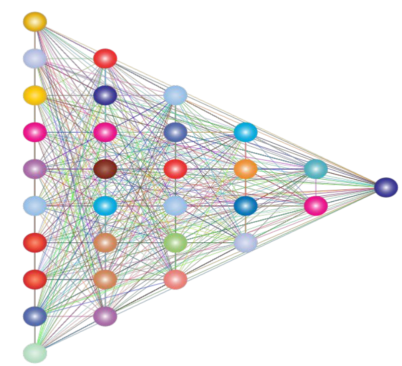
Convolutional (CNN) NN



Markov Chain (MCNN) NN

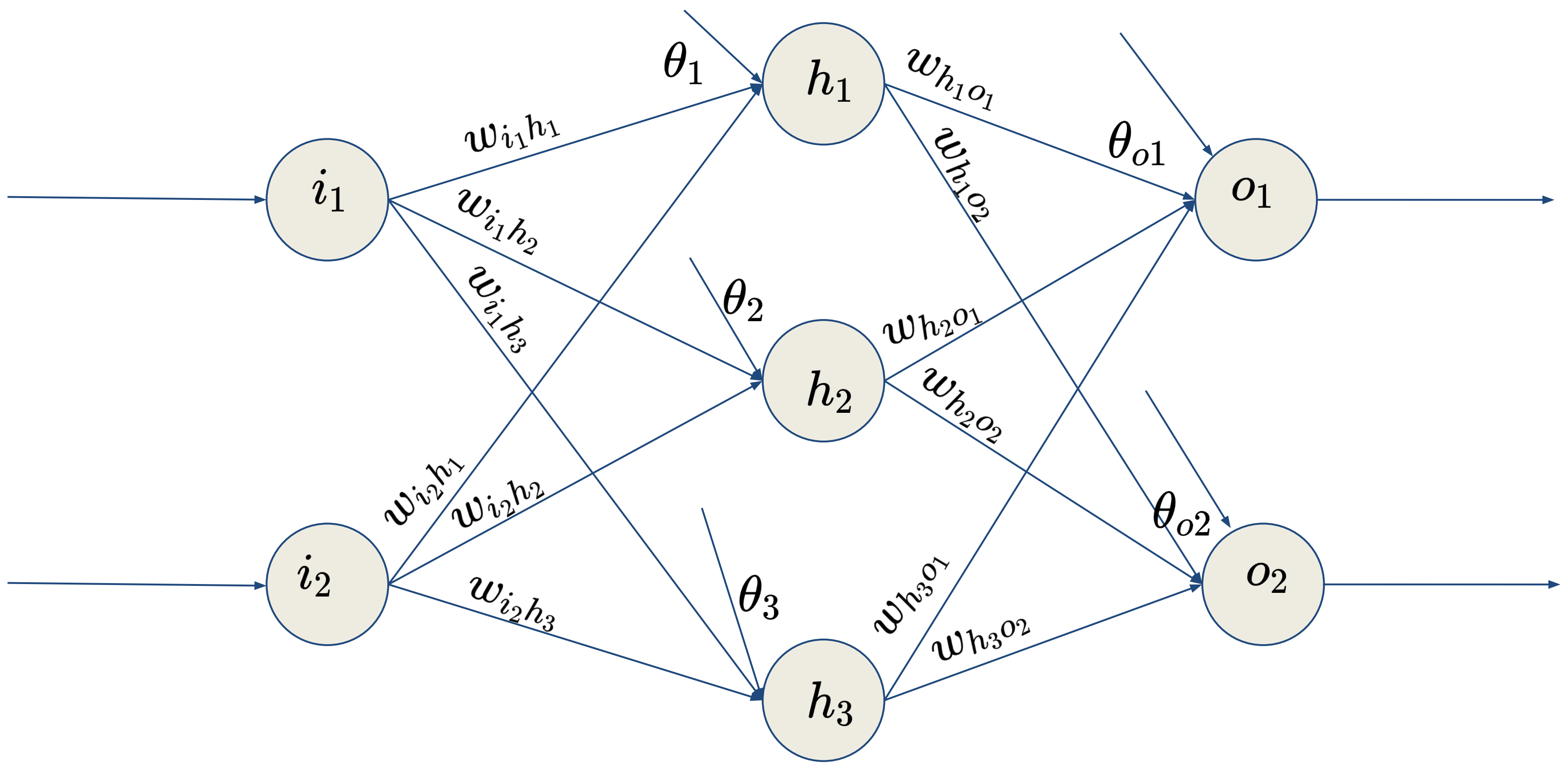


Modular (MNN) NN



Deep Neural Network

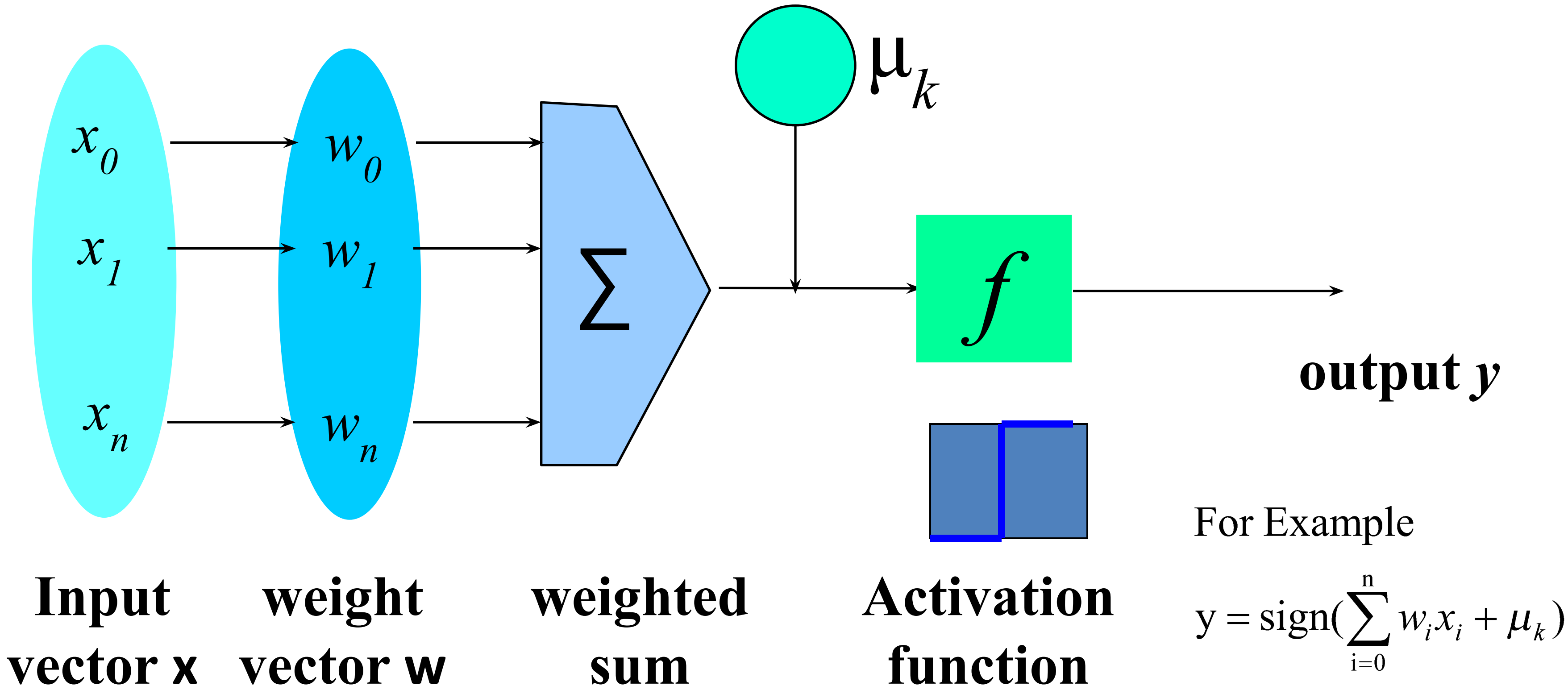
Multi-layer Perceptron



Multi-layer Perceptron NN

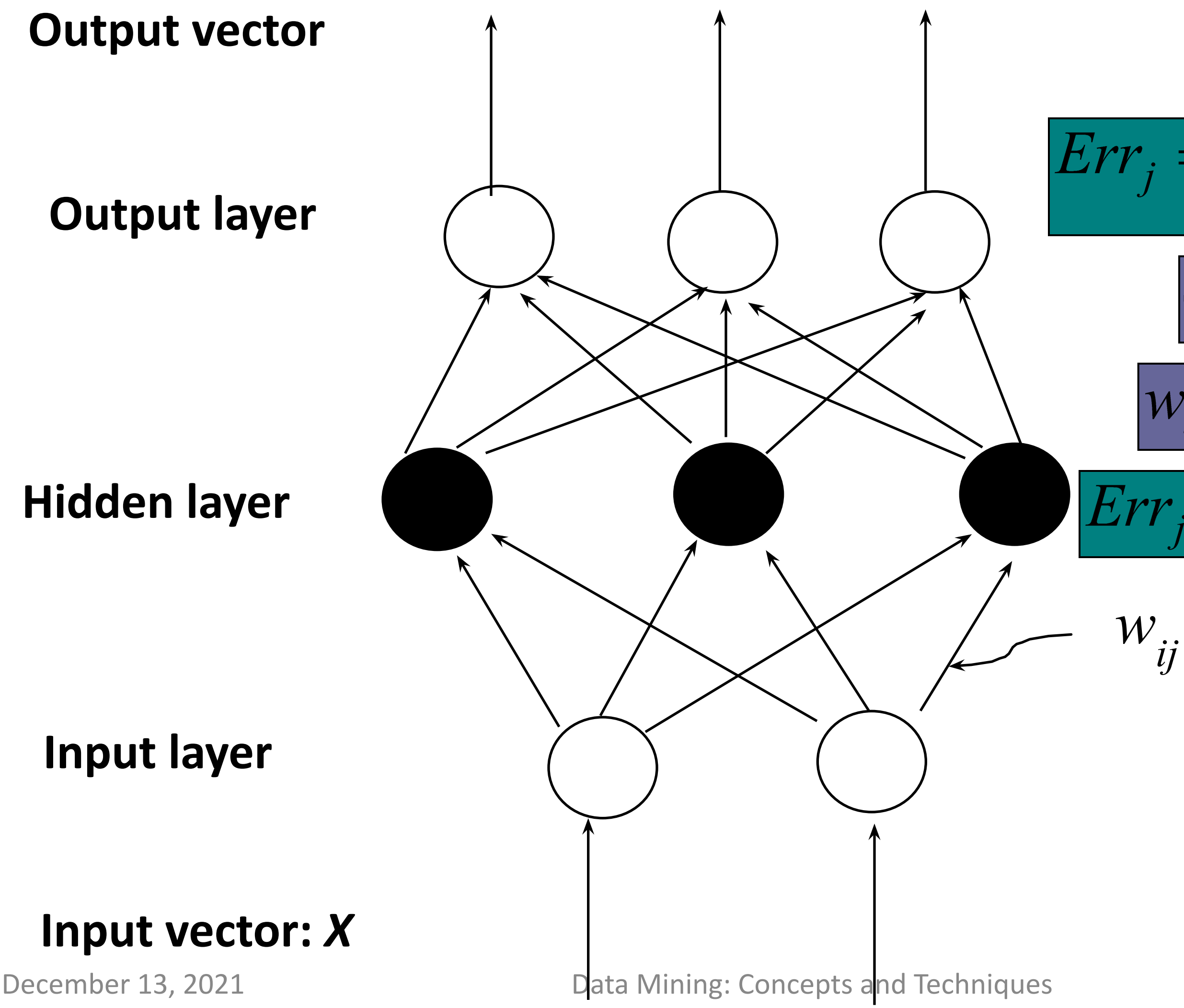
- The most useful type of neural network
- A perceptron is a single neuron model
- Investigates how simply biological brains can be modeled to solve difficult computational tasks
- Develop robust algorithms and data structures
- Power of the ability to learn the representation in training data
- Neurons are the building block
- Neurons are arranged into network of neurons
- Two layer learning and predicting methodologies:
  - Feed Forward
  - Backpropagation
- Activation Function needed on neuron

A Neuron (= a perceptron)



- The *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping

A Multi-Layer Feed-Forward Neural Network



$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

$$\theta_j = \theta_j + (l)Err_j$$

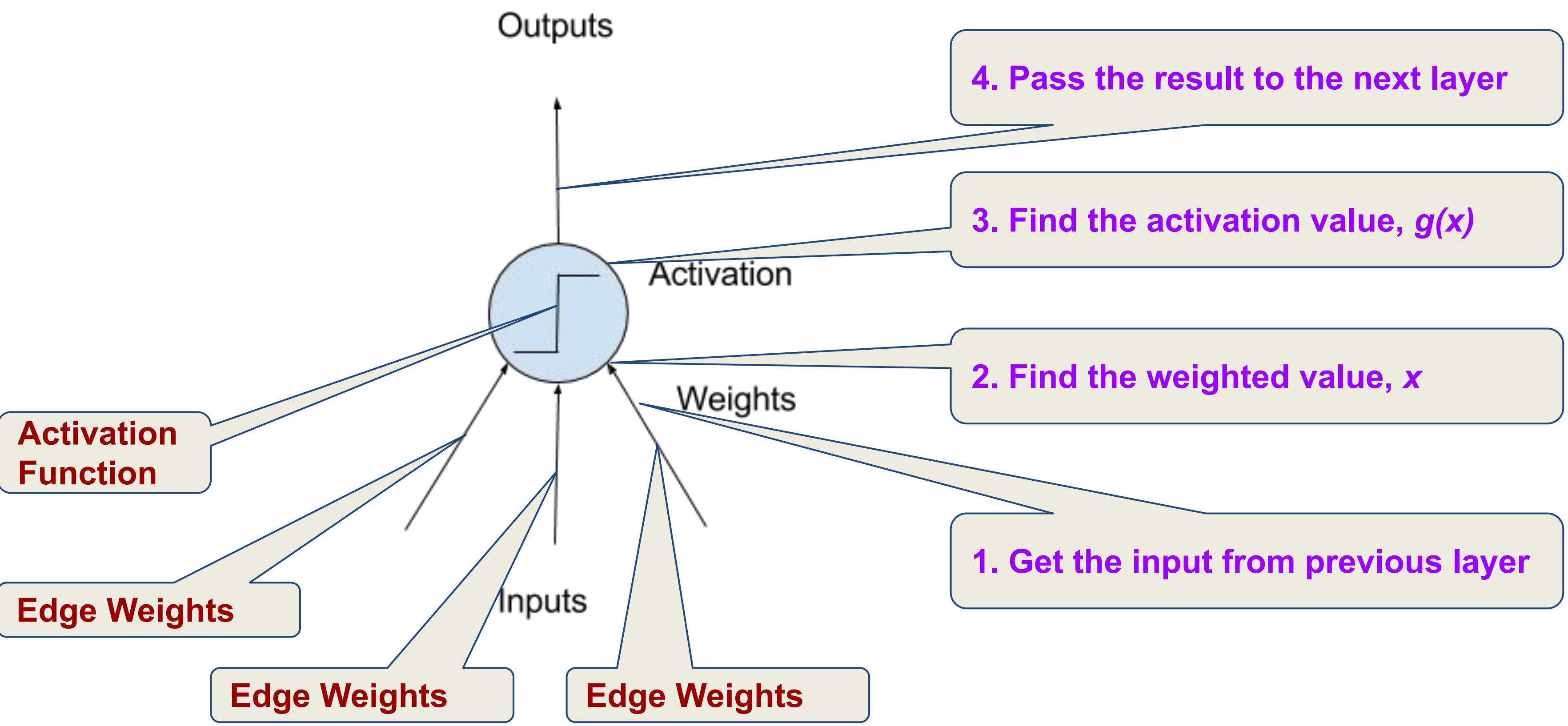
$$w_{ij} = w_{ij} + (l)Err_j O_i$$

$$Err_j = O_j(1-O_j)(T_j - O_j)$$

$$O_j = \frac{1}{1+e^{-I_j}}$$

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

Neuron



Outputs
4. Pass the result to the next layer
3. Find the activation value, *g(x)*
Activation
2. Find the weighted value, *x*
Weights
Activation Function
1. Get the input from previous layer
Edge Weights
Inputs
Edge Weights
Edge Weights

Defining a Network Topology

- First decide the **network topology:** # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalizing the input values for each attribute measured in the training tuples to [0.0—1.0]
- One **input** unit per domain value, each initialized to 0
- **Output**, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a *different network topology* or a *different set of initial weights*

How A Multi-Layer Neural Network Works?

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward** in that none of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights (to small random #s) and biases in the network
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

Backpropagation and Interpretability

- Efficiency of backpropagation: Each epoch (one interation through the training set) takes O(|D| * *w*), with |D| tuples and *w* weights, but # of epochs can be exponential to n, the number of inputs, in the worst case
- Rule extraction from networks: network pruning
  - Simplify the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers
- Sensitivity analysis: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules

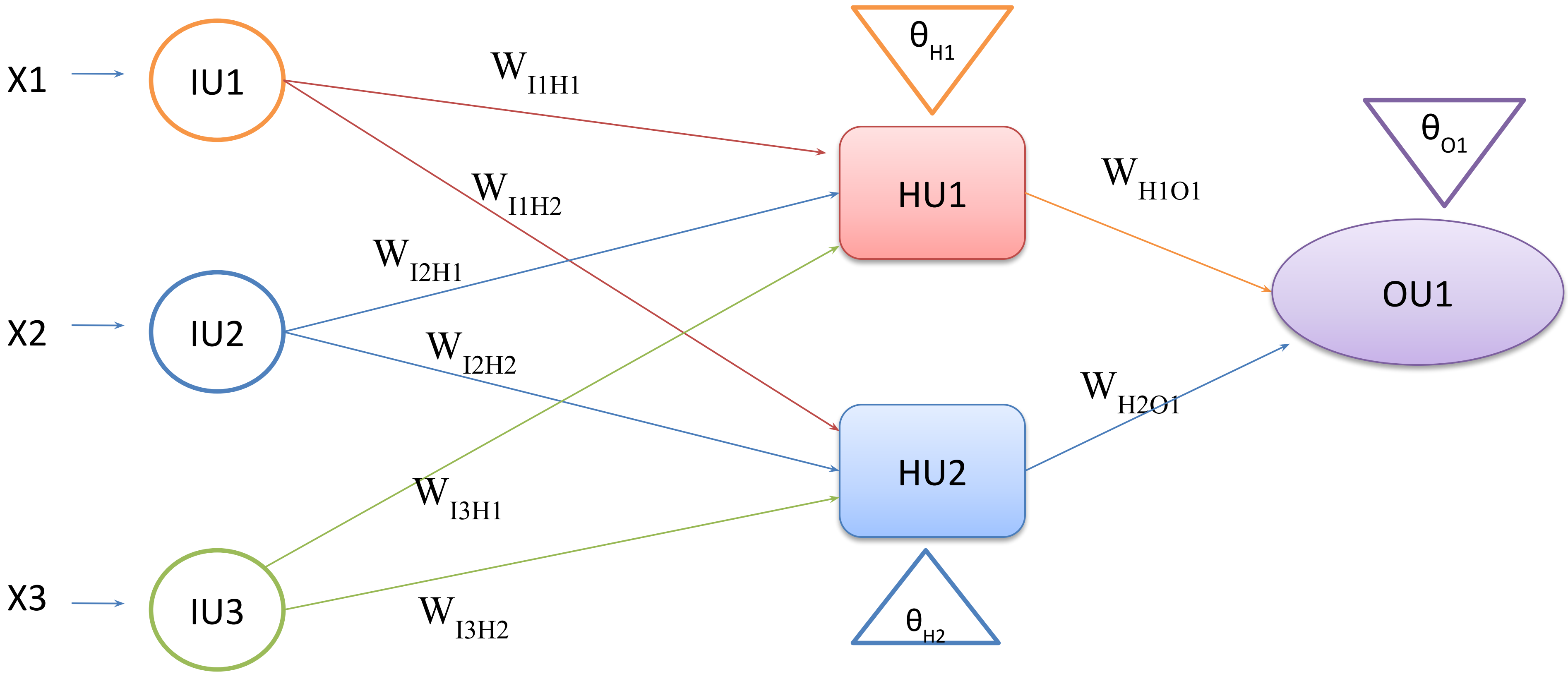
Classification by Backpropagation

- Backpropagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

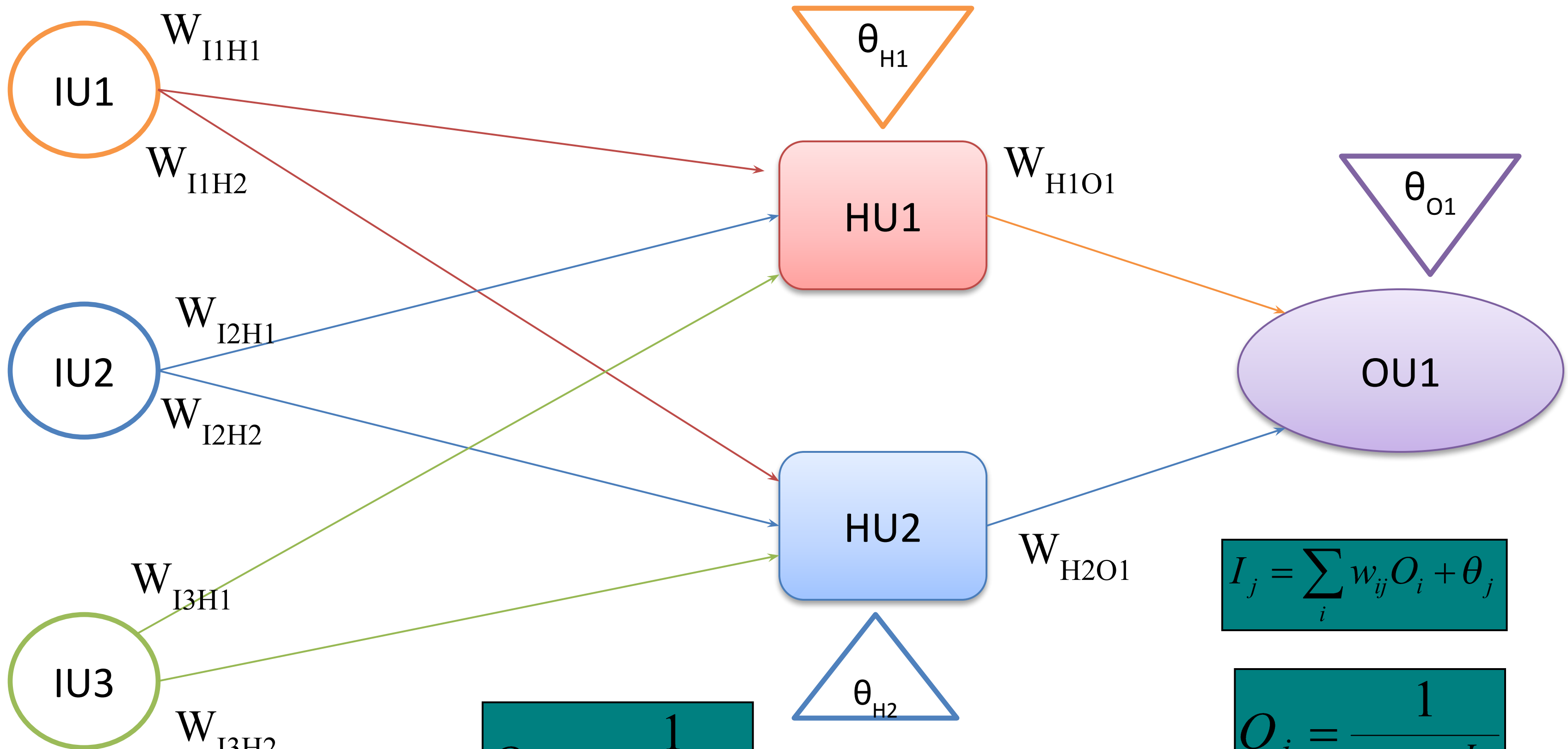
Neural Network as a Classifier

- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or ``structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of ``hidden units" in the network
- Strength
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs and outputs
  - Successful on a wide array of real-world data
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks

Working approach



Working approach



$$I_j = \sum_i w_{ij} O_i + \theta_j$$

$$O_j = \frac{1}{1+e^{-I_j}}$$

$$Err_j = O_j(1-O_j)(T_j - O_j)$$

$$w_{ij} = w_{ij} + (l) Err_j O_i$$

$$\theta_j = \theta_j + (l) Err_j$$

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

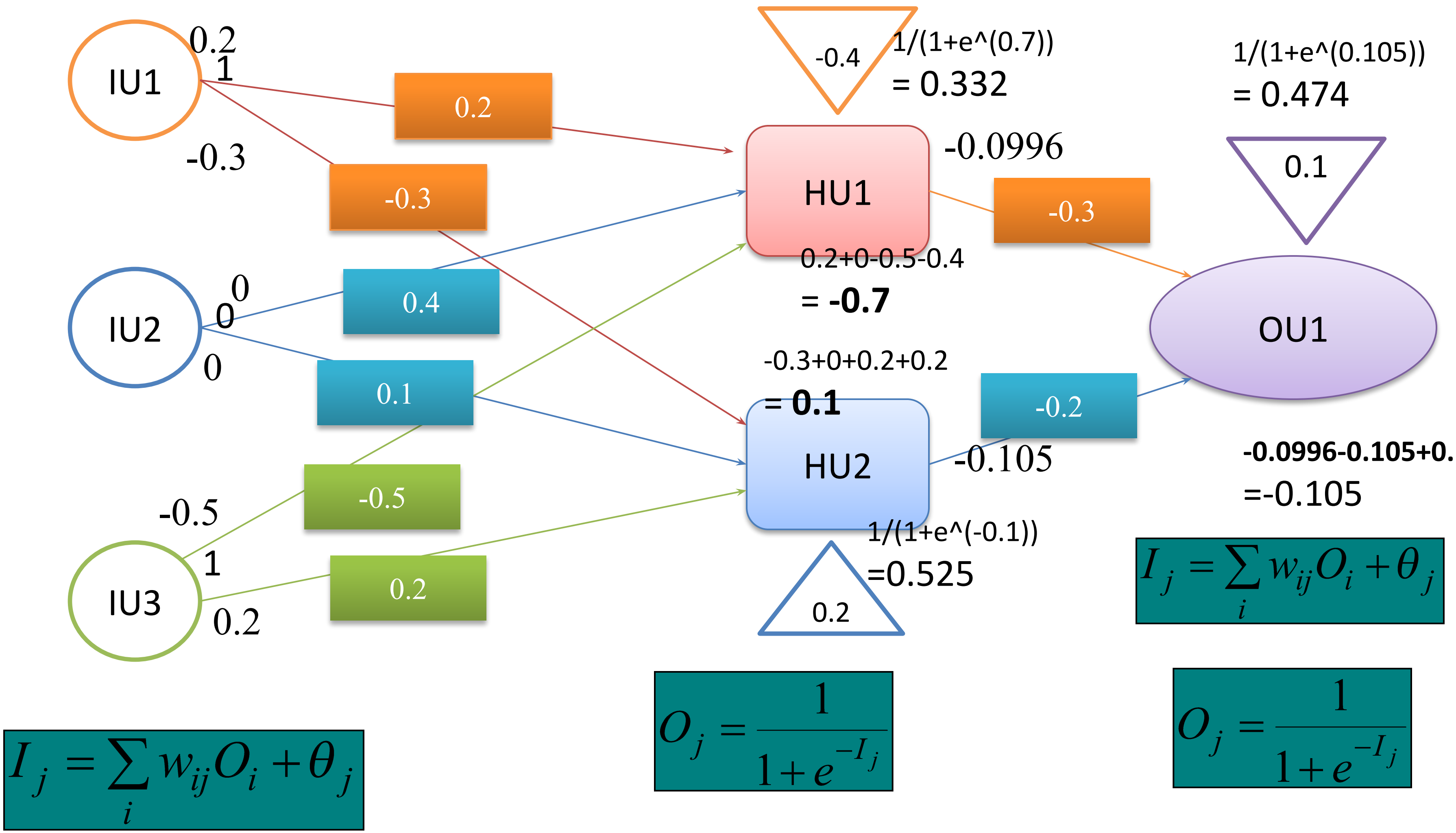
$$O_j = \frac{1}{1+e^{-I_j}}$$

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

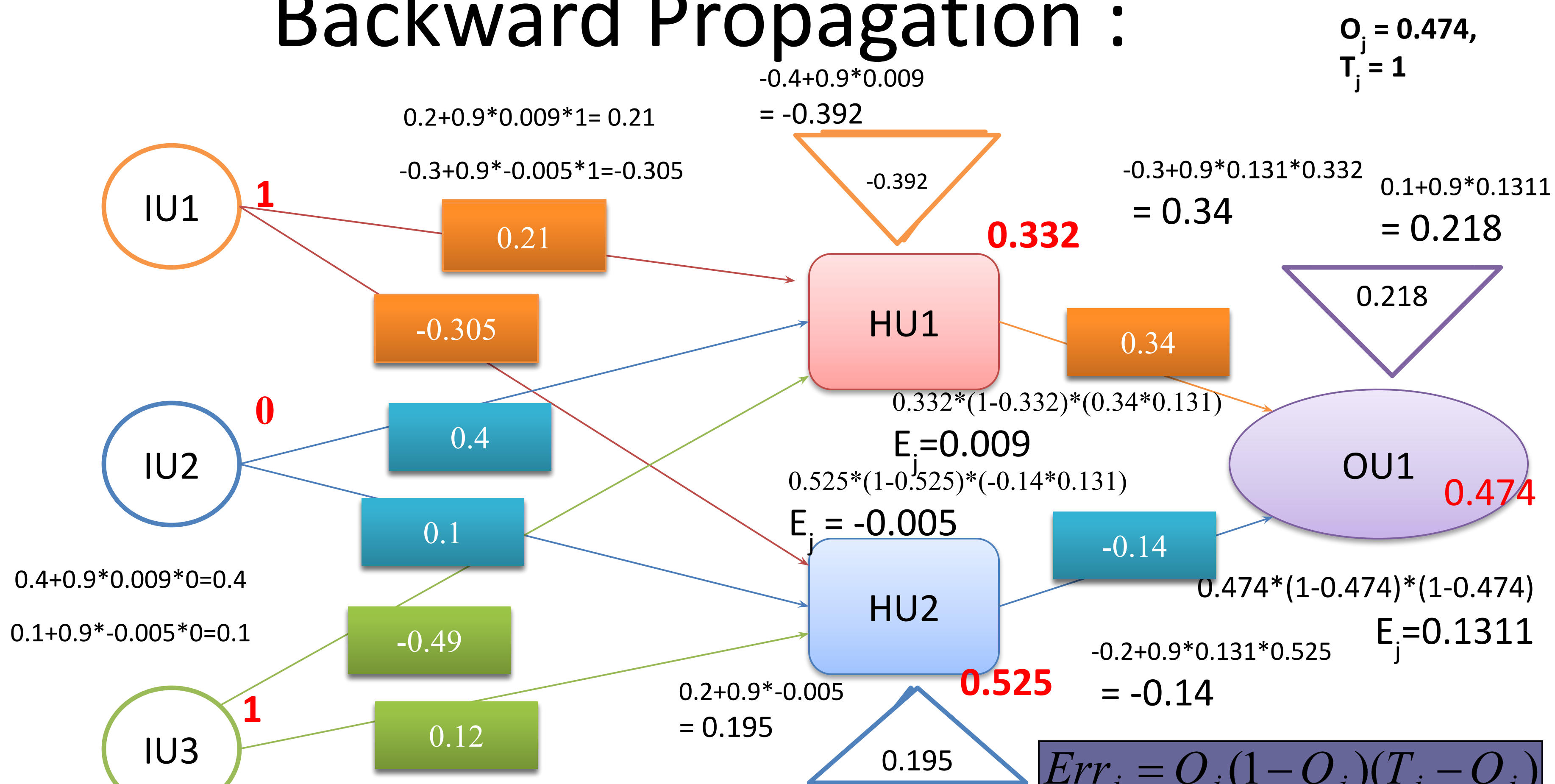
$$w_{ij} = w_{ij} + (l) Err_j O_i$$

$$\theta_j = \theta_j + (l) Err_j$$

Forward Propagation :



Backward Propagation :



$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

$$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$$

l = 0.9

$$w_{ij} = w_{ij} + (l)Err_j O_i$$

$$w_{ij} = w_{ij} + (l)Err_j O_i$$

-0.5+0.9*0.009*1=-0.49

0.2+0.9*-0.005*1=0.12

$$\theta_j = \theta_j + (l)Err_j$$

$$\theta_j = \theta_j + (l)Err_j$$

Conclusions

References