

Artificial Neural network

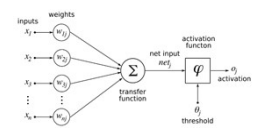
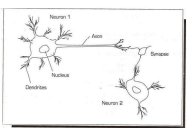
ARTIFICIAL NEURAL NETWORK

Dr. Nibaran Das
Department of Computer Science and engineering

Artificial Neural network

DEFINITION

- An artificial neural network is
 - A massively parallel distributed system,
 - An adaptive system,
 - having a natural property of storing experiential knowledge and making it available for use
- Artificial Neural networks resemble with brain in two aspects:
 - Knowledge is acquired by the network through its learning process
 - Interneuron connection strengths known as synaptic weights are used to store the knowledge

Source: [1] Artificial Neural Networks: Foundations, Principles, and Applications

Artificial Neural network

FUNCTIONING OF AN ARTIFICIAL NEURAL NETWORK

- A Complex Problem is divided into several Small and Simple Calculations without intervention of user.
- These Small and Simple Calculations are processed by a number of CPUs, often called processing elements/neurons/nodes.
- The activity of these neurons are combined to form the output of the Complex Problem.

Artificial Neural network

BENEFITS OF ARTIFICIAL NEURAL NETWORKS

- **Nonlinearity** : To model a physical phenomenon which is in general nonlinear
- **Adaptivity** : To cope with change in environment
- **Evidential response** : To justify a decision
- **Massive parallelism** : To enable faster computation
- **Robustness** : Ability to handle missing, confusing and/or noisy data
- **Fault tolerance** : Ability to work, at least to some extent, even if in component failure
- **Input-output mapping** : Ability to model an arbitrary input-output mapping
- **Optimality** : As regards to error in performance
- **VLSI implementability** : Suited for VLSI implementation

Artificial Neural network **MAIN CONCERN OF AN ARTIFICIAL NEURAL NETWORK**

Architecture :

- Connectivity between nodes, Characteristics/activities of nodes

Learning :

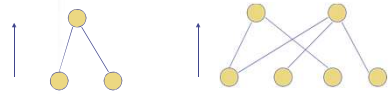
- Way of acquiring knowledge or way of determining weights
- Depending on problem, we select/design a model with a specific architecture and learning algorithm.

Artificial Neural network **ARCHITECTURE**

It is such that

- it can solve a problem for which it is designed and
- local computation is possible

Examples :



Artificial Neural network **VARIOUS ARCHITECTURES**

The manner in which neurons are organized
Various network architectures

- Single-layer feedforward networks (e.g., Perceptron)
- Multilayer feedforward networks (e.g., Multilayer Perceptron)
- Recurrent networks (e.g., Hopfield network)
- Lattice structures (e.g., Kohonen network)
- Combination of above

Artificial Neural network **LEARNING**

In the context of artificial neural networks, learning is a process of

- Adapting free parameters of a neural network
- Through a continuous stimulation by the environment in which network is embedded
- Type of learning is determined by the manner in which the parameter changes take place

Artificial Neural network

TAXONOMY OF LEARNING PROCESS

- Learning paradigm
 - Supervised
 - Unsupervised
 - Reinforcement
- Learning algorithms
 - Error-correction learning
 - Boltzmann learning
 - Hebbian learning
 - Competitive learning

Artificial Neural network

SUPERVISED AND UNSUPERVISED LEARNING

Supervised Learning: Learning proceeds in presence of teacher signal

Unsupervised Learning: Learning proceeds in absence of teacher signal

Artificial Neural network

REINFORCEMENT LEARNING

- Coined by Marvin Minsky in 1961
- Complete supervised information not available, i.e., situation not instructive
- Only partial supervised information available, e.g., output right/wrong. Situation evaluative
- Learning with "critic" as opposed to learning with "teacher"
- Through "reward" and "punishment" Involving some source of randomness in the network

Artificial Neural network

THORNDIKE'S LAW OF EFFECT (1911)

Related to Animal learning in Psychology

If an action taken by a learning system is followed by a satisfactory state of affairs, then the tendency of the system to produce that particular action is strengthened or reinforced. Otherwise, the action is weakened.

Artificial Neural network

TYPES OF REINFORCEMENT LEARNING

Non-associative: learning system selects a single optimal action, e.g., function optimization under GA framework

Associative: learning system associates different actions with different stimuli, e.g., studied under neural network framework, Thorndike's law of effect

Artificial Neural network

SOME WELL-KNOWN MODELS

- Based on Supervised learning : Multilayer Perceptron
- Radial Basis Function network Based on Unsupervised learning :
- Kohonen's Self-organizing Feature Map (SOM)
- Hopfield model

Artificial Neural network

CONNECTIONIST MODELS

Consider humans

- Neuron switching time ~ 0.001 second
- Number of neurons $\sim 10^{10}$
- Connections per neuron $\sim 10^{4-5}$
- Scene recognition time ~ 0.1 second
- 100 inference step does not seem like enough

must use lots of parallel computation!

Properties of artificial neural nets (ANNs):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

15

Artificial Neural network

WHEN TO CONSIDER NEURAL NETWORKS

- Input is high-dimensional discrete or real-valued (e.g., raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is *unimportant*

Examples:

- Speech phoneme recognition [Waibel]
- Image classification [Kanade, Baluja, Rowley]
- Financial prediction

16

Artificial Neural network

ALVINN DRIVES 70 MPH ON HIGHWAYS

30x32 Sensor Input Retina

4 Hidden Units

Sharp Left, Straight Ahead, Sharp Right

17

Artificial Neural network

PERCEPTRON

$$\sigma = \sum_{i=1}^n w_i x_i$$

$$\sigma = \begin{cases} 1 & \text{if } \sum_{i=1}^n w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Sometimes we will use simpler vector notation :

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

18

Artificial Neural network

ACTIVATION FUNCTIONS

threshold

linear

piece-wise linear

sigmoid

19

Artificial Neural network

DECISION SURFACE OF PERCEPTRON

Represents some useful functions

- What weights represent $g(x_1, x_2) = \text{AND}(x_1, x_2)$?

But some functions not representable

- e.g., not linearly separable
- therefore, we will want networks of these ...

20

Artificial Neural network

PERCEPTRON TRAINING RULE

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta (t - o) x_i$$

- $t = c(\bar{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called learning rate

Can prove it will converge

- If training data is linearly separable
- and η is sufficiently small

21

Artificial Neural network

THE PERCEPTRON

- The operation of Rosenblatt's perceptron is based on the McCulloch and Pitts neuron model. The model consists of a linear combiner followed by a hard limiter.
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and -1 if it is negative.

© Negnevitsky, Pearson Education, 2011

22

Artificial Neural network

LINEAR SEPARABILITY IN THE PERCEPTRONS

- The aim of the perceptron is to classify inputs, x_1, x_2, \dots, x_n , into one of two classes, say A_1 and A_2 .
- In the case of an elementary perceptron, the n -dimensional space is divided by a *hyperplane* into two decision regions. The hyperplane is defined by the *linearly separable* function:

$$\sum_{i=1}^n x_i w_i - \theta = 0$$

© Negnevitsky, Pearson Education, 2011

23

Artificial Neural network

LINEAR SEPARABILITY IN THE PERCEPTRONS

(a) Two-input perceptron. $x_1 w_1 + x_2 w_2 - \theta = 0$

(b) Three-input perceptron. $x_1 w_1 + x_2 w_2 + x_3 w_3 - \theta = 0$

© Negnevitsky, Pearson Education, 2011

24

Artificial Neural network

HOW DOES THE PERCEPTRON LEARN ITS CLASSIFICATION TASKS?

This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range $[-0.5, 0.5]$, and then updated to obtain the output consistent with the training examples.

25

Artificial Neural network

- If at iteration p , the actual output is $Y(p)$ and the desired output is $Y_d(p)$, then the error is given by:

$$e(p) = Y_d(p) - Y(p) \quad \blacksquare \text{ where } p = 1, 2, 3, \dots$$

Iteration p here refers to the p th training example presented to the perceptron.

- If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

26

Artificial Neural network

THE PERCEPTRON LEARNING RULE

$$w_i(p+1) = w_i(p) + \alpha \cdot x_i(p) \cdot e(p)$$

where $p = 1, 2, 3, \dots$
 α is the learning rate, a positive constant less than unity.

The perceptron learning rule was first proposed by Rosenblatt in 1960. Using this rule we can derive the perceptron training algorithm for classification tasks.

© Negnevitsky, Pearson Education, 2011

27

Artificial Neural network

PERCEPTRON'S TRAINING ALGORITHM

Step 1: Initialisation
 Set initial weights w_1, w_2, \dots, w_n and threshold θ to random numbers in the range $[-0.5, 0.5]$.

Step 2: Activation
 Activate the perceptron by applying inputs $x_1(p), x_2(p), \dots, x_n(p)$ and desired output $Y_d(p)$.
 Calculate the actual output at iteration $p = 1$

$$Y(p) = \text{step} \left[\sum_{i=1}^n x_i(p) w_i(p) - \theta \right]$$

where n is the number of the perceptron inputs, and step is a step activation function.

© Negnevitsky, Pearson Education, 2011

28

Artificial Neural network **PERCEPTRON'S TRAINING ALGORITHM (CONTINUED)**

Step 3: Weight training
Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration p .
The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

Step 4: Iteration
Increase iteration p by one, go back to *Step 2* and repeat the process until convergence.

© Negnevitsky, Pearson Education, 2011

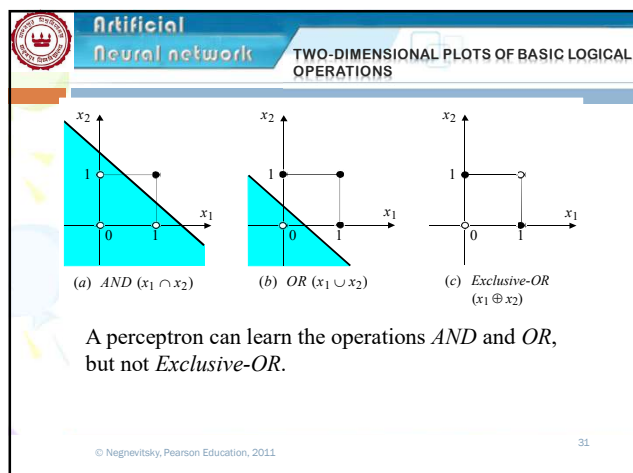
Artificial Neural network **PERCEPTRON'S TRAINING ALGORITHM (CONTINUED)**

EXAMPLE OF PERCEPTRON LEARNING: THE LOGICAL OPERATION AND

Threshold: $\theta = 0.2$;
learning rate: $\alpha = 0.1$

Epoch	Inputs		Desired output Y_d	Initial weights		Actual output Y	Error e	Final weights	
	x_1	x_2		w_1	w_2			w_1	w_2
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	1	1	1	0.2	-0.1	0	0	0.3	-0.1
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	-1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	-1	0.1	0.0
	1	1	1	0.1	0.0	0	0	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	-1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

© Negnev



Artificial Neural network **PERCEPTRON CONVERGENCE THEOREM**

The algorithm converges to the correct classification

- if the training data is linearly separable
- and η is sufficiently small

➤ If two classes of vectors X_1 and X_2 are linearly separable, the application of the perceptron training algorithm will eventually result in a weight vector \mathbf{w}_0 , such that \mathbf{w}_0 defines a TLU whose decision hyper-plane separates X_1 and X_2 (Rosenblatt 1962).

➤ Solution \mathbf{w}_0 is not unique, since if $\mathbf{w}_0 \cdot \mathbf{x} = 0$ defines a hyper-plane, so does $\mathbf{w}'_0 = k \mathbf{w}_0$.

Artificial Neural network GRADIENT DESCENT

To understand, consider simple *linear unit*, where

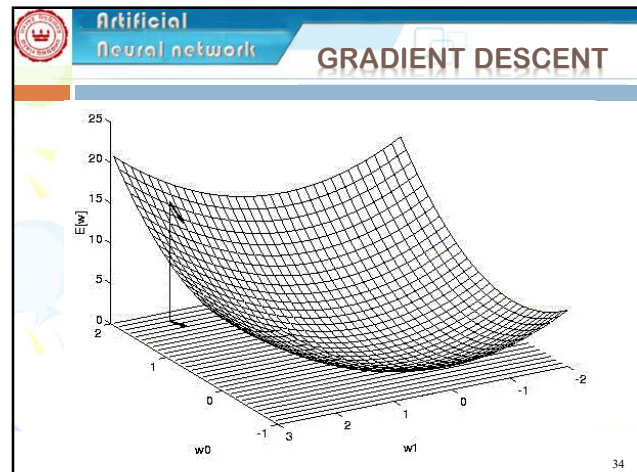
$$o = w_0 + w_1x_1 + \dots + w_nx_n$$

Idea : learn w_i 's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is the set of training examples

33



Artificial Neural network GRADIENT DESCENT

Gradient $\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]^T$

Training rule: $\Delta w_i = -\eta \nabla E[\vec{w}]$

i.e., $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$

35

Artificial Neural network GRADIENT DESCENT

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ \frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d) (-x_{i,d}) \end{aligned}$$

36

Artificial Neural network **GRADIENT DESCENT**

GRADIENT – DESCENT (*training _ examples, η*)

Each training examples is a pair of the form $\langle \vec{x}, t \rangle$, where \vec{x} is the vector of input values and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training _ examples*, do
 - * Input the instance \vec{x} and compute output o
 - * For each linear unit weight w_i , do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$
 - For each linear unit weight w_i , do

$$w_i \leftarrow w_i + \Delta w_i$$

37

Artificial Neural network **INCREMENTAL (STOCHASTIC) GRADIENT DESCENT**

Batch mode Gradient Descent:

Do until satisfied:

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Incremental mode Gradient Descent:

Do until satisfied:

- For each training example d in D
 1. Compute the gradient $\nabla E_d[\vec{w}]$
 2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily closely if η made small enough

38

Artificial Neural network **PERCEPTRON VS. GRADIENT DESCENT RULE**

- perceptron rule

$$w'_i = w_i + \alpha (t^p - y^p) x_i^p$$
 derived from manipulation of decision surface.
- gradient descent rule

$$w'_i = w_i + \alpha (t^p - y^p) x_i^p$$
 derived from minimization of error function

$$E[w_1, \dots, w_n] = \frac{1}{2} \sum_p (t^p - y^p)^2$$
 by means of gradient descent.

Where is the big difference?

Artificial Neural network **SUMMARY**

Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule uses gradient descent

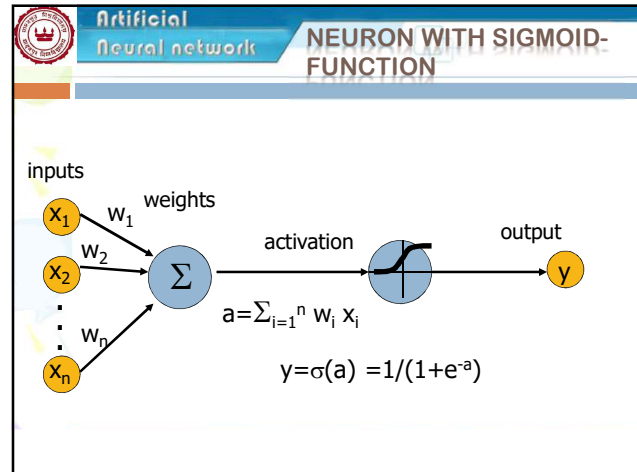
- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

40

Artificial Neural network

PRESENTATION OF TRAINING EXAMPLES

- Presenting all training examples once to the ANN is called an *epoch*.
- In incremental stochastic gradient descent training examples can be presented in
 - ❑ Fixed order (1,2,3,...,M)
 - ❑ Randomly permuted order (5,2,7,...,3)
 - ❑ Completely random (4,1,7,1,5,4,.....)



Artificial Neural network

SIGMOID UNIT

$x_0 = -1$ w_0 $a = \sum_{i=0}^n w_i x_i$ $y = \sigma(a) = 1/(1+e^{-a})$

$\sigma(x)$ is the sigmoid function: $1/(1+e^{-x})$
 $d\sigma(x)/dx = \sigma(x) (1 - \sigma(x))$

Derive gradient decent rules to train:

- one sigmoid function
- $\partial E / \partial w_i = -\sum_p (t^p - y) y (1 - y) x_i^p$
- Multilayer networks of sigmoid units backpropagation:

Artificial Neural network

GRADIENT DESCENT RULE FOR SIGMOID OUTPUT FUNCTION

sigmoid σ

$E^p[w_1, \dots, w_n] = 1/2 (t^p - y^p)^2$

$\partial E^p / \partial w_i = \partial / \partial w_i 1/2 (t^p - y^p)^2$

$= \partial / \partial w_i 1/2 (t^p - \sigma(\sum_i w_i x_i^p))^2$

$= (t^p - y^p) \sigma'(\sum_i w_i x_i^p) (-x_i^p)$

for $y = \sigma(a) = 1/(1+e^{-a})$

$\sigma'(a) = e^{-a} / (1+e^{-a})^2 = \sigma(a) (1 - \sigma(a))$

$w'_i = w_i + \Delta w_i = w_i + \alpha y(1-y)(t^p - y^p) x_i^p$

Artificial Neural network

GRADIENT DESCENT LEARNING RULE

$$\Delta w_{ji} = \alpha y_j^p (1 - y_j^p) (t_j^p - y_j^p) x_i^p$$

learning rate α

derivative of activation function $y_j^p (1 - y_j^p)$

error δ_j of post-synaptic neuron $(t_j^p - y_j^p)$

activation of pre-synaptic neuron x_i^p

Artificial Neural network

LEARNING WITH HIDDEN UNITS

- Networks without hidden units are very limited in the input-output mappings they can model.
 - ❑ More layers of linear units do not help. Its still linear.
 - ❑ Fixed output non-linearities are not enough
- We need multiple layers of adaptive non-linear hidden units. This gives us a universal approximator. But how can we train such nets?
 - ❑ We need an efficient way of adapting **all** the weights, not just the last layer. This is hard. Learning the weights going into hidden units is equivalent to learning features.
 - ❑ Nobody is telling us directly what hidden units should do.

Artificial Neural network

LEARNING BY PERTURBING WEIGHTS

Randomly perturb one weight and see if it improves performance. If so, save the change.

- ❑ **Very inefficient.** We need to do multiple forward passes on a representative set of training data just to change one weight.
- ❑ Towards the end of learning, large weight perturbations will nearly always make things **worse**.

We could randomly perturb all the weights in parallel and correlate the performance gain with the weight changes.

- ❑ Not any better because we need lots of trials to "see" the effect of changing one weight through the noise created by all the others.

output units

hidden units

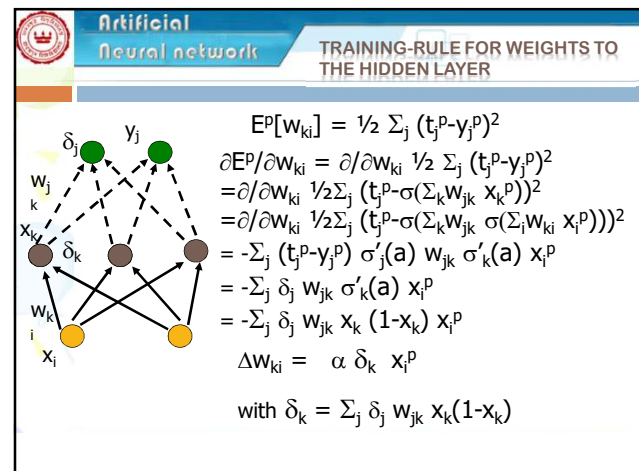
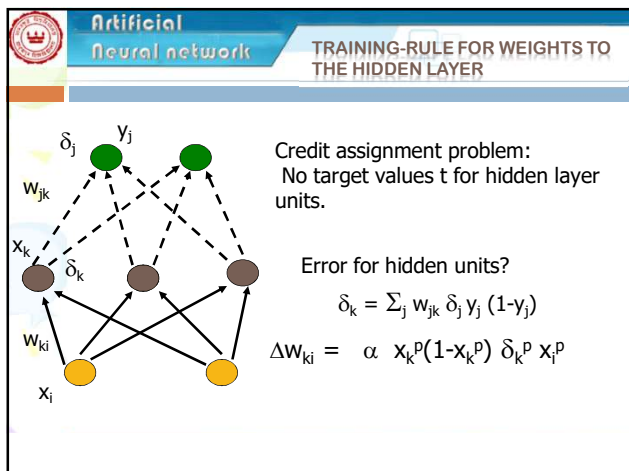
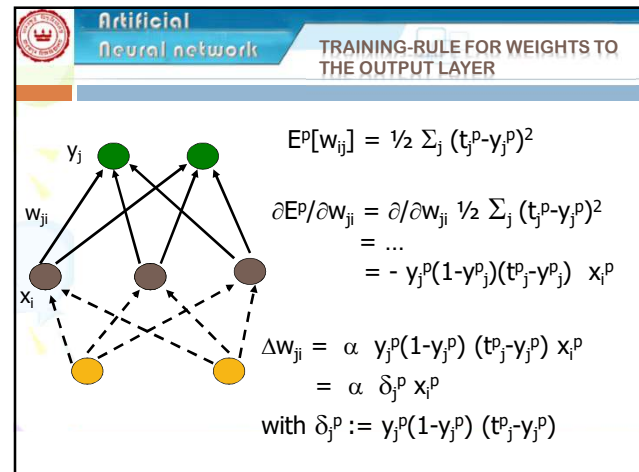
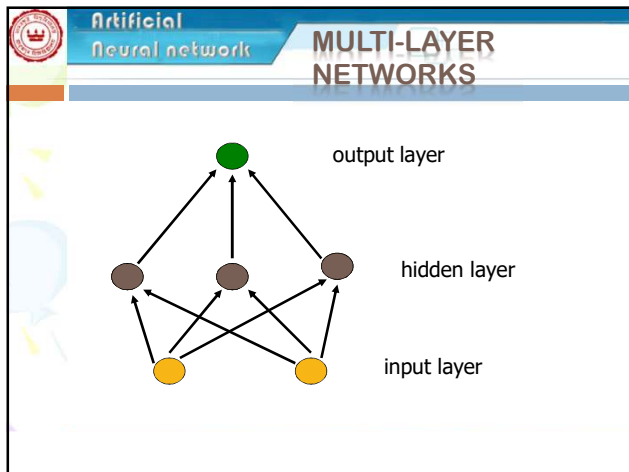
input units

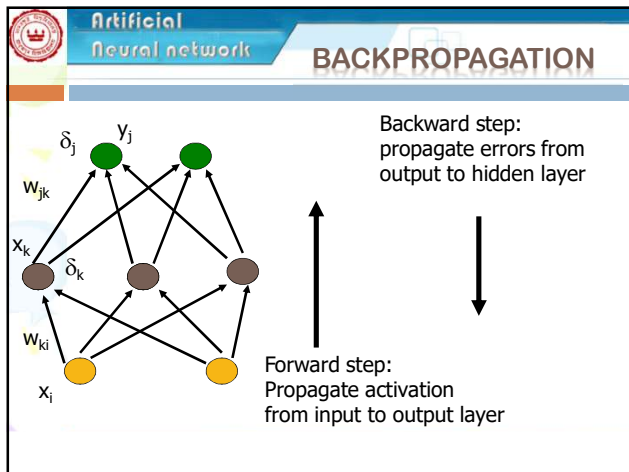
Learning the hidden to output weights is **easy**. Learning the input to hidden weights is **hard**.

Artificial Neural network

THE IDEA BEHIND BACKPROPAGATION

- We don't know what the hidden units ought to do, but we can compute how fast the error changes as we change a hidden activity.
 - ❑ Instead of using desired activities to train the hidden units, use **error derivatives w.r.t. hidden activities**.
 - ❑ Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.
 - ❑ We can compute error derivatives for **all** the hidden units efficiently.
 - ❑ Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.





Artificial Neural network

BACKPROPAGATION ALGORITHM

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - ❑ For each training example $\langle (x_1, \dots, x_n), t \rangle$ Do
 - Input the instance (x_1, \dots, x_n) to the network and compute the network outputs y_k
 - For each output unit k
 - * $\delta_k = y_k(1-y_k)(t_k - y_k)$
 - For each hidden unit h
 - * $\delta_h = y_h(1-y_h) \sum_k w_{h,k} \delta_k$
 - For each network weight $w_{i,j}$ Do
 - $w_{i,j} = w_{i,j} + \Delta w_{i,j}$ where
 - $\Delta w_{i,j} = \eta \delta_j x_{i,j}$

Artificial Neural network

BACKPROPAGATION

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
 - in practice often works well (can be invoked multiple times with different initial weights)
- Often include weight *momentum* term
 - $\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$
- Minimizes error training examples
 - ❑ Will it generalize well to unseen instances (over-fitting)?
- Training can be slow typical 1000-10000 iterations
- Using network after training is fast

Artificial Neural network

CONVERGENCE OF BACKPROP

Gradient descent to some local minimum perhaps not global minimum

- Add momentum term: $\Delta w_{ki}(n)$
 - ❑ $\Delta w_{ki}(n) = \alpha \delta_k(n) x_i(n) + \lambda \Delta w_{ki}(n-1)$ with $\lambda \in [0,1]$
- Stochastic gradient descent
- Train multiple nets with different initial weights
- Nature of convergence
 - Initialize weights near zero
 - Therefore, initial networks near-linear
 - Increasingly non-linear functions possible as training progresses