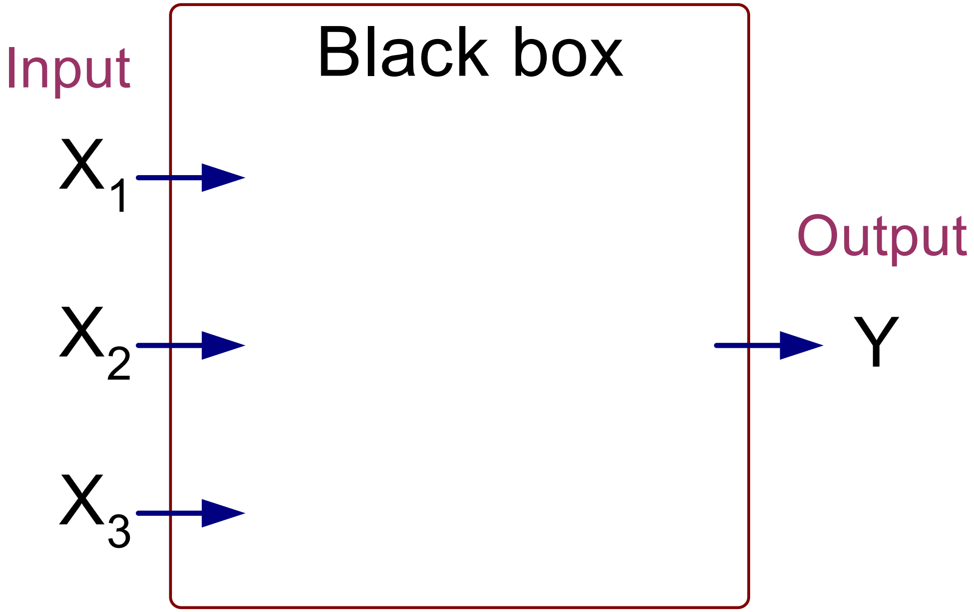
Artificial Neural Networks
CS 584: Data Mining

Prof. Sanmay Das
George Mason University

Based on the notes of
Tan, Steinbach, Karpatne, Kumar

Artificial Neural Networks (ANN)

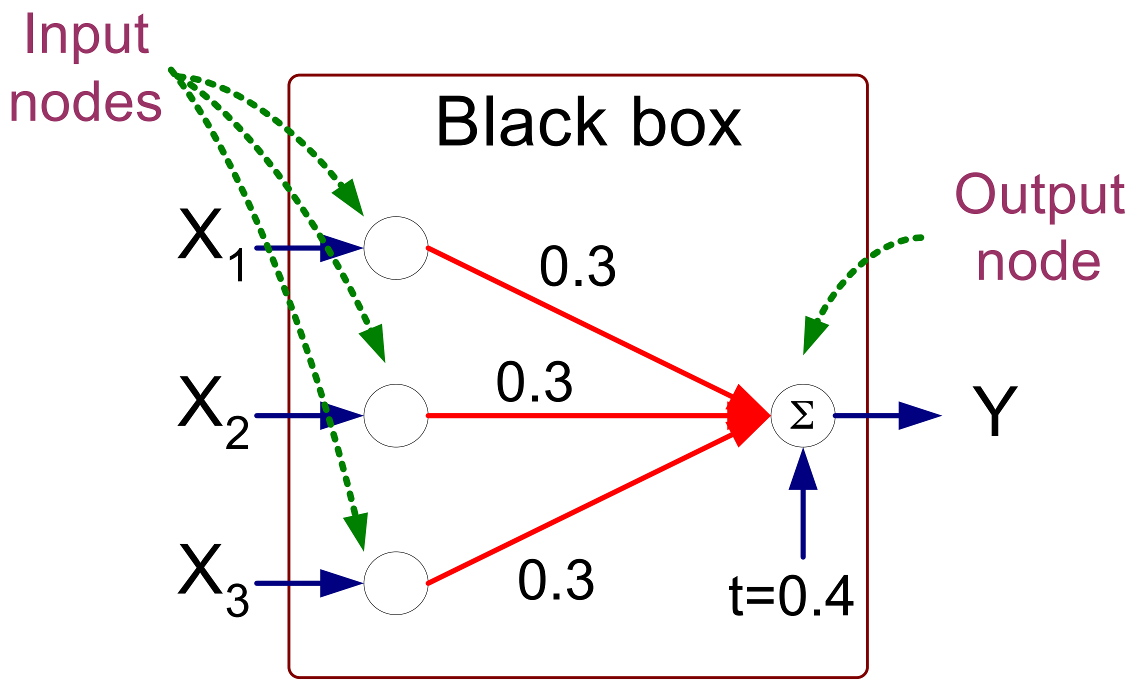
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

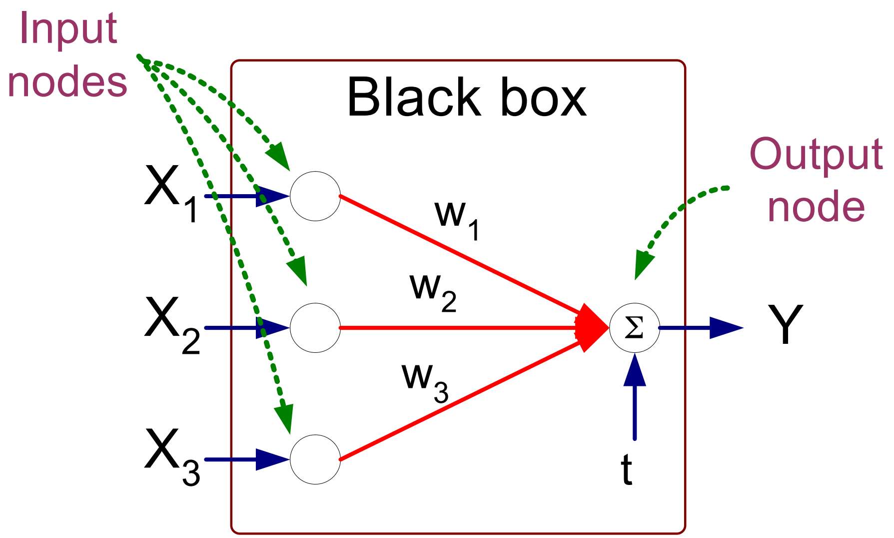
$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links

- Output node sums up each of its input value according to the weights of its links

- Compare output node against some threshold t

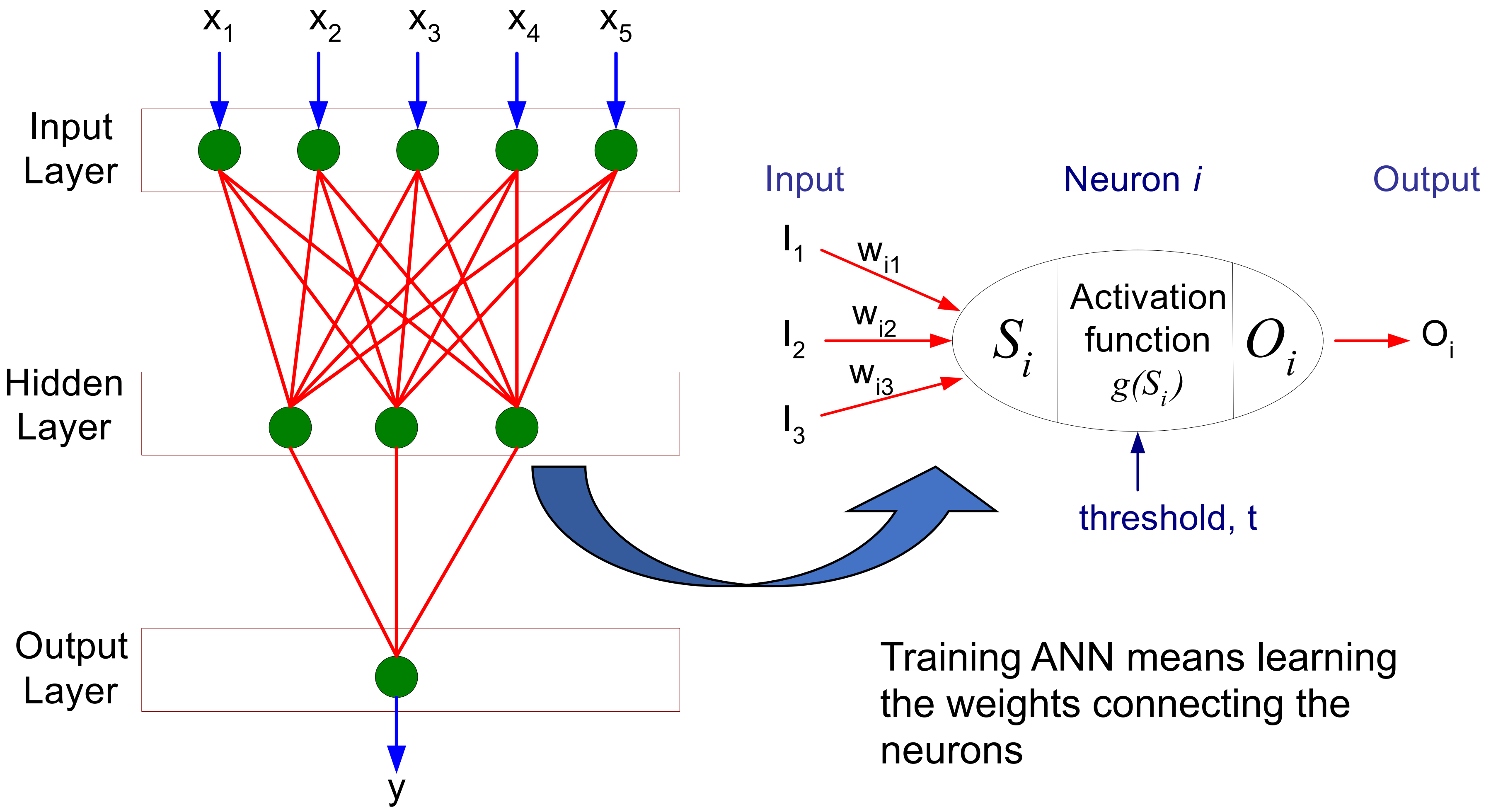


Perceptron Model

$$Y = sign(\sum_{i=1}^{d} w_i X_i - t)$$

$$= sign(\sum_{i=0}^{d} w_i X_i)$$

General Structure of ANN



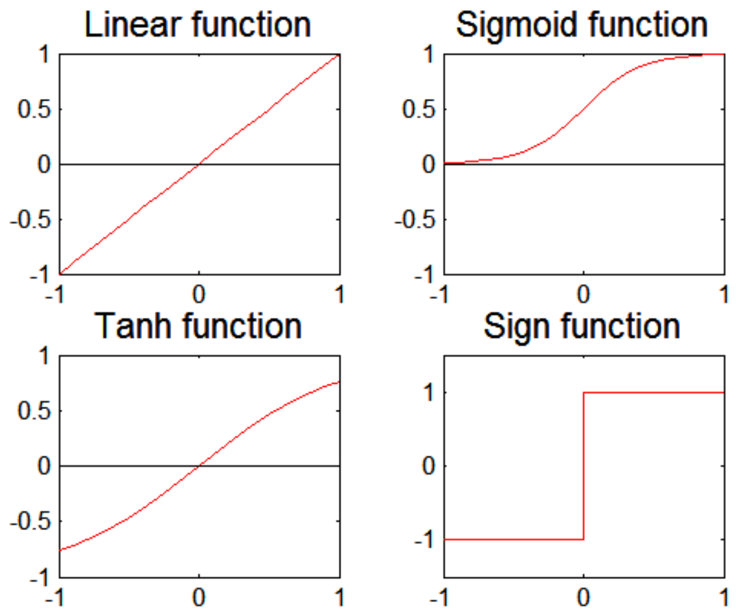
Training ANN means learning the weights connecting the neurons

Artificial Neural Networks (ANN)

- Various types of neural network topology
  - single-layered network (perceptron) versus multi-layered network
  - Feed-forward versus recurrent network

- Various types of activation functions (f)

$$Y = f\left(\sum_i w_i X_i\right)$$



- In deep networks: often ReLU (rectified linear units / hinge function)

Perceptron

- Single layer network
  - Contains only input and output nodes

- Activation function: f = sign(w•x)

- Applying model is straightforward

$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

  - $X_1 = 1$, $X_2 = 0$, $X_3 = 1$ => y = sign(0.2) = 1

Perceptron Learning Rule

- Initialize the weights (w_0, w_1, …, w_d)
- Repeat
  - For each training example ($\mathbf{x}_i$, y_i)
    - Compute $f(\mathbf{w}, \mathbf{x}_i)$
    - Update the weights: $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \lambda[y_i - f(\mathbf{w}^{(k)}, \mathbf{x}_i)]\mathbf{x}_i$
- Until stopping condition is met

Perceptron Learning Rule

- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i \ \ ; \ \ \lambda\text{: learning rate}$$

- Intuition:
  - Update weight based on error: $e = [y_i - f(\mathbf{w}^{(k)}, \mathbf{x}_i)]$
  - If y=f(x,w), e=0: no update needed
  - If y>f(x,w), e=2: weight must be increased so that f(x,w) will increase
  - If y<f(x,w), e=-2: weight must be decreased so that f(x,w) will decrease

Example of Perceptron Learning

$$\lambda = 0.1$$

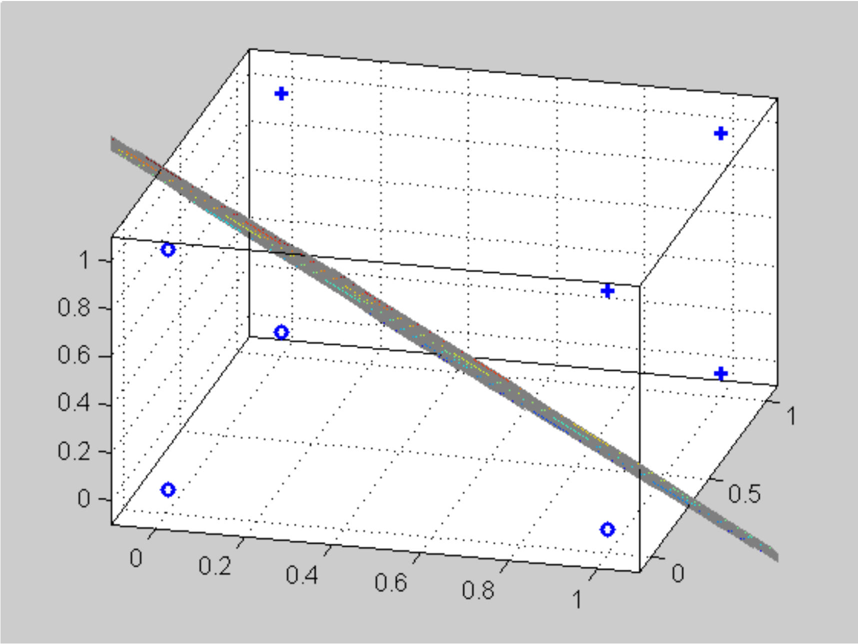
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2

Perceptron Learning Rule

- Since f(w,x) is a linear combination of input variables, decision boundary is linear



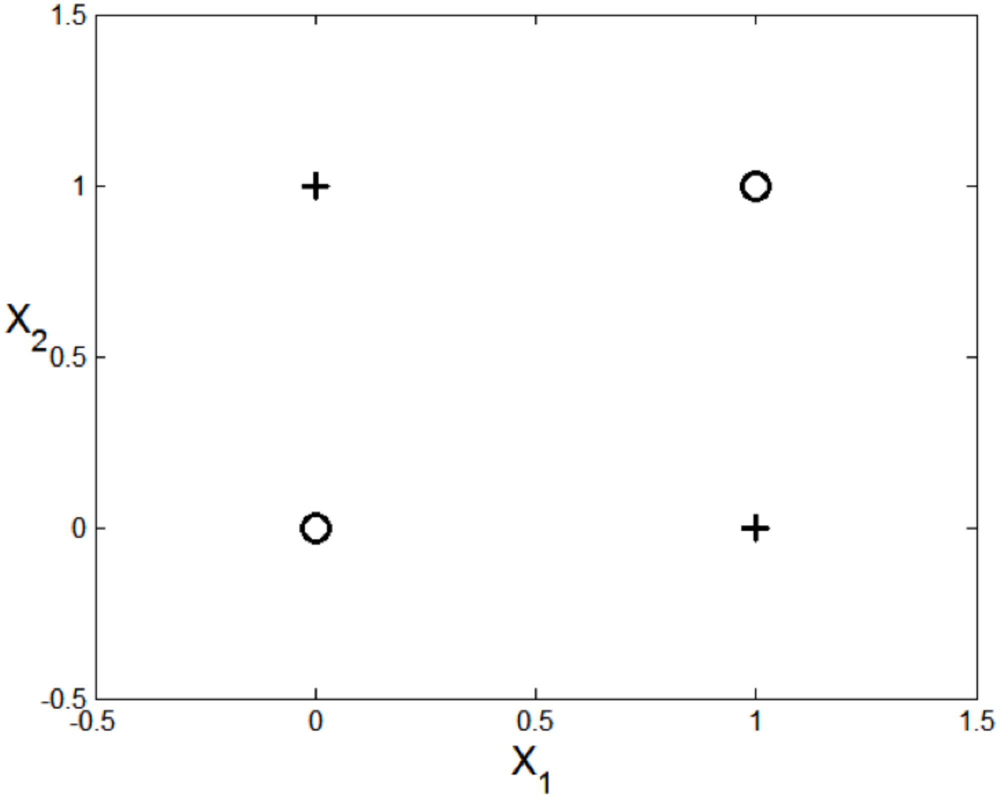
- For nonlinearly separable problems, perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly

Nonlinearly Separable Data

$$y = x_1 \oplus x_2$$

x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1

XOR Data



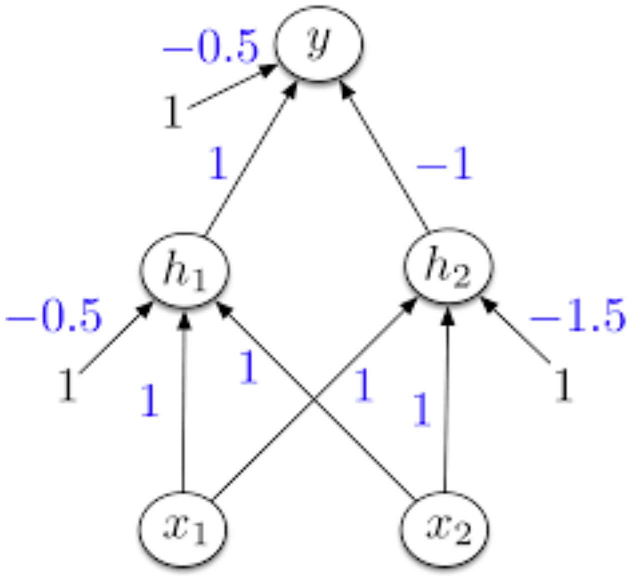
Multilayer Neural Network

- Hidden layers
  - intermediary layers between input & output layers

- More general activation functions (sigmoid, linear, etc)

Multi-layer Neural Network

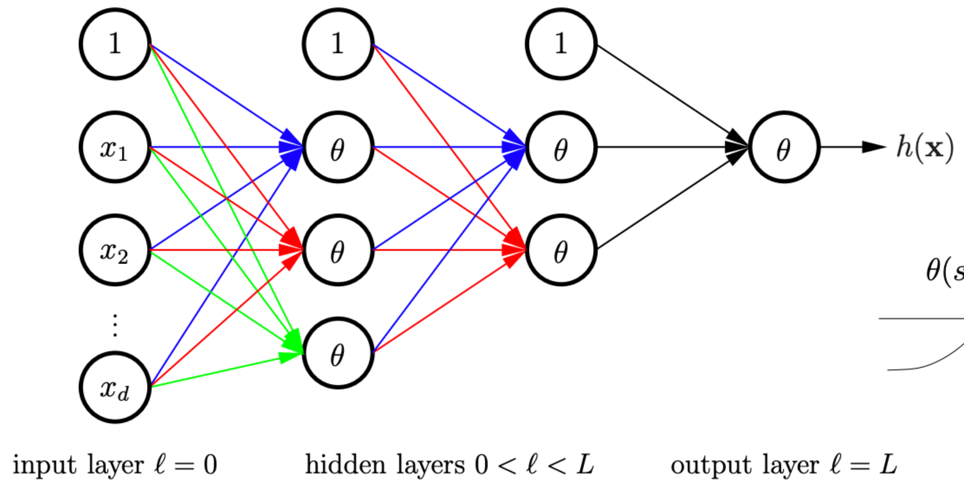
- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces. e.g. XOR

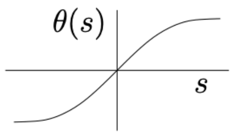


Learning Multi-layer Neural Network

- Can we apply the perceptron learning rule to each node, including hidden nodes?
  - Perceptron learning rule computes error term e = y-f(w,x) and updates weights accordingly
    - Problem: how to determine the true value of y for hidden nodes?
  - Approximate error in hidden nodes by error in the output nodes
    - Problem:
      - Not clear how adjustment in the hidden nodes affect overall error
      - No guarantee of convergence to optimal solution

With a fixed architecture, how do we learn the weight matrices?





Step 1: Forward Prop

$$x^{(l)} = \begin{bmatrix} 1 \\ \theta(s^{(l)}) \end{bmatrix}$$

$$s^{(l)} = (w^{(l)})^{\mathrm{T}} x^{(l-1)}$$

$$\text{Training error (SoS) } E_{\text{in}}(w) = \frac{1}{N}\sum_{n=1}^{N}(X_n^{(L)} - y_n)^2)$$

$s^{(l)}$: $d^{(l)}$ dim. input vector

$x^{(l)}$: $d^{(l)} + 1$ dim. output vector (includes the 1)

$w^{(l)}$: $(d^{(l-1)} + 1) \times d^{(l)}$ dim. weight matrix

$w^{(l+1)}$: $(d^{(l)} + 1) \times d^{(l+1)}$ dim. weight matrix

Step 2: BackProp

Basic idea: Use gradient descent to find a local min of E_{in}.
$w(t+1) = w(t) - \eta \nabla E_{\text{in}}(w(t))$
Consider partial derivative of error of a single data point w.r.t. weights at a layer $\frac{\partial e}{\partial w^{(l)}}$
Sensitivity vector $\delta^{(l)} = \frac{\partial e}{\partial s^{(l)}}$
Then, $\frac{\partial e}{\partial w^{(l)}} = x^{(l-1)}(\delta^{(l)})^{\mathrm{T}}$
So, partial derivatives w.r.t. weights coming into this layer can be found using activations and previous layer and the sensitivity vector

Turns out that $\delta^{(l)}$ can be computed as:

$$\theta'(s^{(l)}) \otimes \left[w^{(l+1)}\delta^{(l+1)}\right]_1^{d^{(l)}}$$

Design Issues for Single-Layer ANNs

- Number of nodes in input layer
  - One input node per binary/continuous attribute
  - k or $\log_2$ k nodes for each categorical attribute with k values
- Number of nodes in output layer
  - One output for binary class problem
  - k or $\log_2$ k nodes for k-class problem
- Number of nodes in hidden layer
- Initial weights and biases

Recent Noteworthy Developments in ANN

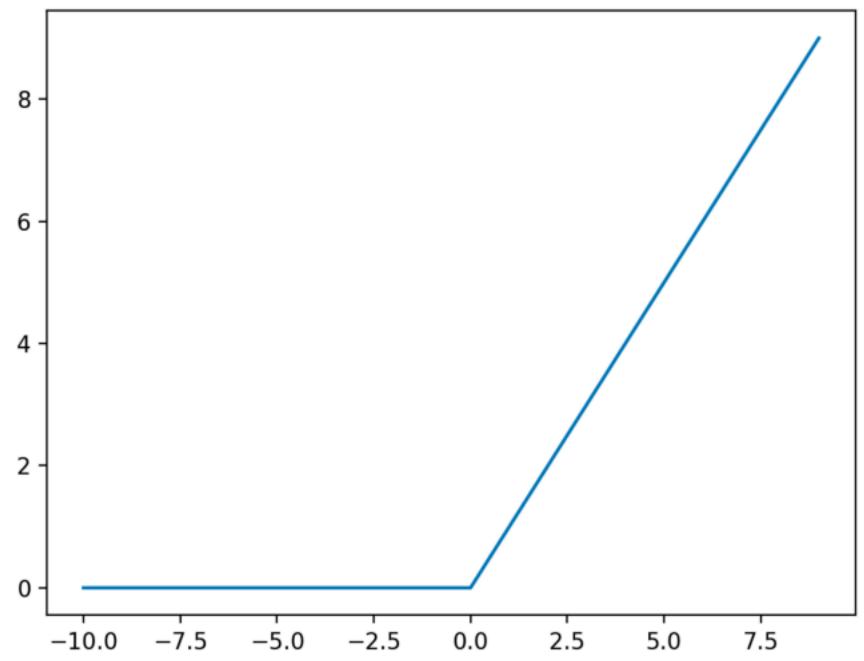
- Use in deep learning and unsupervised feature learning
  - Seek to automatically learn a good representation of the input from unlabeled data
- Google Brain project
  - Learned the concept of a ‘cat’ by looking at unlabeled pictures from YouTube
  - One billion connection network

Deep Neural Networks

- Involve a large number of hidden layers
- Can represent features at multiple levels of abstraction
- Often require fewer nodes per layer to achieve generalization performance similar to shallow networks
- Deep networks have become the technique of choice for complex problems such as vision and language processing

Rectified Linear Units (ReLU)

- $\theta(z) = \max(0, z)$
- Not differentiable at 0!
  - Typically take either the left or the right derivative (0 or 1)
  - Gradient based optimization is already subject to numerical error
  - GD still performs well



Deep Nets: Challenges and Solutions

- Challenges
  - Slow convergence
  - Sensitivity to initial values of model parameters
  - The larger number of nodes makes deep networks susceptible to overfitting

- Solutions
  - Large training data sets
  - Advances in computational power, e.g., GPUs
  - Algorithmic advances
    - New architectures and activation units
    - Better parameter and hyper-parameter selection
    - Regularization

Deep Learning Characteristics

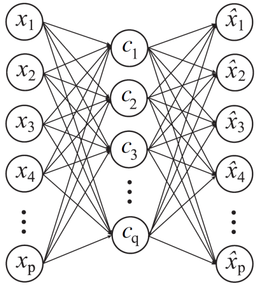
- Pre-training allow deep learning models to reuse previous learning.
  - The learned parameters of the original task are used as initial parameter choices for the target task
  - Particularly useful when the target application has a smaller number of labeled training instances than the one used for pre-training

- Deep learning techniques for regularization help in reducing the model complexity
  - Lower model complexity promotes good generalization performance
  - The dropout method is one regularization approach
  - Regularization is especially important when we have
    - high-dimensional data
    - a small number of training labels
    - the classification problem is inherently difficult.

Deep Learning Characteristics …

- Using an autoencoder for pretraining can
  - Help eliminate irrelevant attributes
  - Reduce the impact of redundant attributes.

Single-layer
Autoencoder



- Deep models, could find inferior and locally optimal solutions
  - Various regularization techniques have been proposed and used
  - Example: Skip connections, dropout

- Specialized ANN architectures have been designed to handle various data sets.
  - Convolutional Neural Networks (CNN) handle two-dimensional gridded data and are used for image processing
  - Recurrent Neural Network handles sequences and are used to process speech and language
  - Transformers add an attention mechanism and have been very successful for language