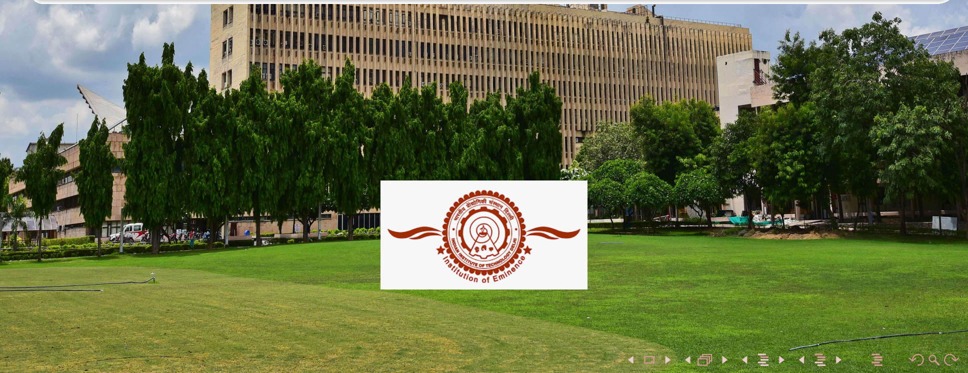
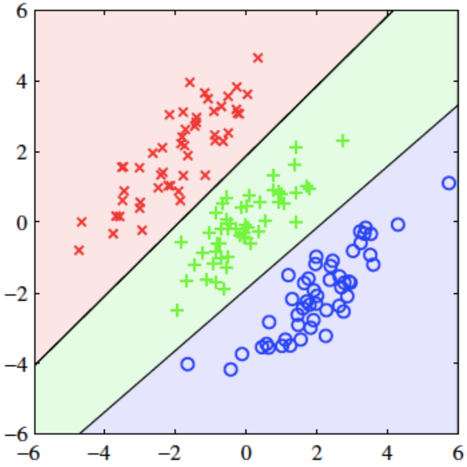
EVIDYA DSML
Multi Layer Perceptron

Hariprasad Kodamana, Manojkumar Ramteke, Agam Gupta
IIT DELHI



Linear decision boundaries



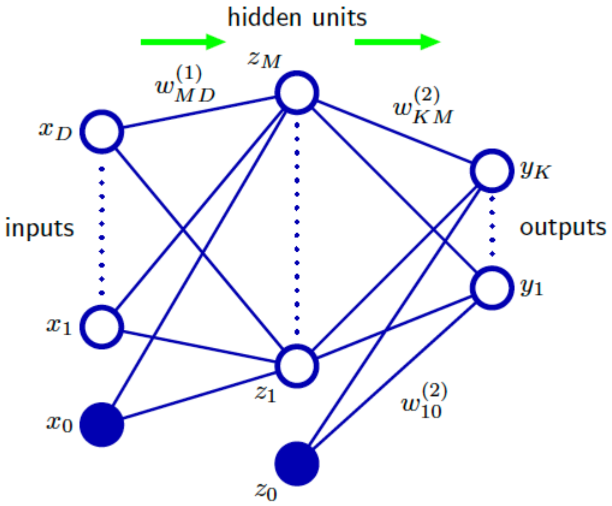
Overview of Presentation

1 Multilayer perceptrons

Multilayer perceptrons

- The perceptron can only be expected to handle problems that are linearly separable.
- To tackle more complicated (nonlinear) situations, we can increase the set of perceptrons in multiple layers
- The additional layers are called'hidden' layers between the output and input layers.
- Also popularly called feed-forward neural nets or feed-forward networks
- According to Bishop- "Indeed, it has been used very broadly to cover a wide range of different models, many of which have been the subject of exaggerated claims regarding their biological plausibility."
- The model comprises multiple layers of logistic regression models (with continuous nonlinearities) rather than multiple perceptrons (with discontinuous nonlinearities)

Multilayer perceptrons: structure



Multilayer perceptrons: structure

- Structure of basis functions:

$$y(x, w) = f(\sum_{i=1}^{M} w_j \phi_j(x)) \tag{1}$$

where $f(.)$ is a nonlinear activation function in the case of classification and is the identity in the case of regression, $\phi(x) = x$ if variables are considered directly

Multilayer perceptrons: structure

- Output of first hidden layer:

$$z_j = h(a_j) = h(\sum_{i=1}^{D} w_{ji}^{(1)} x_i) \tag{2}$$

  input variables-$x_1, \dots, x_D$, $j = 1, \dots, M$ -hidden layer units, w_{ji} -weights, a_j - hidden layer activations, h -hidden layer activation function

- Outputs unit activation a_k

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j \tag{3}$$

  $k = 1, ..., K$, K- total number of outputs, $w_{kj}^{(2)}$ -output weights

- Outputs:

$$y_k = \begin{cases} a_k \ \ if \ \ regression \\ \sigma(a_k) \ \ if \ \ classification \end{cases} \tag{4}$$

  $\sigma(.)$ is the output function

Multilayer perceptrons: structure

- Network output y_k:

$$y_k(x, w) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{5}$$

- Deep networks: generally used buzz word if there are multiple hidden layers.
- Credit assignment path : chain of transformations from input to output. Eg. For a feedforward neural network, the depth of the CAP is the number of hidden layers plus one
- Deep network : CAP depth> 2
- CAP of depth 2 has been shown to be a universal approximators

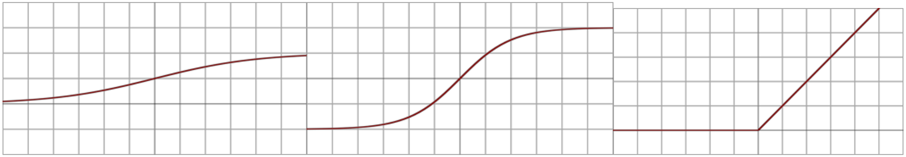
Multilayer perceptrons: activation functions

- Output activation fuction: mostly logistic function

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{6}$$

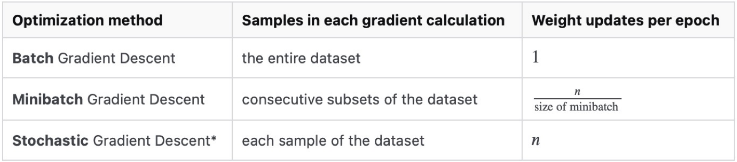
- For hidden layer activation fuction:

$$h(a) = \begin{cases} Logistic([0,1]) - \frac{1}{1+exp(-a)} \\ tanh - ([-1,1]) - \frac{e^a - e^{-a}}{e^a + e^{-a}} \\ ReLU - \begin{cases} 0 \ if \ \ a \leq 0 \\ a \ if \ \ a > 0 \end{cases} \end{cases} \tag{7}$$



Epoch, batch and mini batch

- An epoch is typically one loop over the entire dataset.
- A batch or minibatch refers to equally sized subsets of the dataset over which the gradient is calculated and weights updated.



Optimization method	**Samples in each gradient calculation**	**Weight updates per epoch**
Batch Gradient Descent	the entire dataset	1
Minibatch Gradient Descent	consecutive subsets of the dataset	$\frac{n}{\text{size of minibatch}}$
Stochastic Gradient Descent*	each sample of the dataset	n

-

Back propagation: training multilayer perceptrons

- Shorthand for “the backward propagation of errors”, since an error is computed at the output and distributed backwards throughout the network’s layers
- The multilayer perceptron architecture is sometimes called a backpropagation network
- Involves two simple steps:
    1. In the first stage, the derivatives of the error function with respect to the weights must be evaluated
    2. In the second stage, the derivatives are then used to compute the adjustments to be made to the weights.

Back propagation (contd...)

Notation k indicates output layer, notation j indicates hidden layers

- Error function:

$$E(w) = \sum_{n=1}^{N} E_n(w) \tag{8}$$

- Consider n^{th} data vector, then, for all outputs $y_1, \ldots, y_k$:

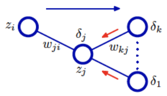
$$E_n(w) = \frac{1}{2} \sum_{k} (y_{nk} - t_{nk})^2 \tag{9}$$

  y_{nk}- k^{th} output of the n^{th} data vector, t_{nk}-corresponding target of y_{nk} (Omitting subscript n for convenience for parameters)

- In a feed-forward network (forward propagation), Output of a general hidden layer: ($z_i = x_i$, if there is only one hidden layer)

$$z_j = h(a_j) = h(\sum_{i=1}^{M} w_{ji} z_i) \tag{10}$$

Back propagation (contd...)



- Gradient calculation for output layer

$$\frac{\partial E_n}{\partial w_{kj}} = \underbrace{\frac{\partial E_n}{\partial a_k}}_{\delta_k} \underbrace{\frac{\partial a_k}{\partial w_{kj}}}_{z_j} \tag{11}$$

- Gradient calculation for hidden layer:

$$\frac{\partial E_n}{\partial w_{ji}} = \underbrace{\frac{\partial E_n}{\partial a_j}}_{\delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i} \tag{12}$$

$$\frac{\partial E_n}{\partial a_j} = \sum_{k=1}^{K} \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{13}$$

Back propagation (Contd..)

Until desired training criterion is met, repeat the following steps

1. Apply an input vector x_n to the network and forward propagate through the network using to find the activations of all the hidden and output units.
2. Perfrom weight update
   - Output layer weight update using GD

$$w_{kj} := w_{kj} - \alpha \frac{\partial E_n}{\partial w_{kj}} \tag{14}$$

   - Hidden layer weight update using GD

$$w_{ij} := w_{ij} - \alpha \frac{\partial E_n}{\partial w_{ji}} \tag{15}$$

Thank you!