Non-linear Hypothesis
Introduction of Multilayer Perceptron

Machine Learning (CS 306)

Instructor: Dr. Moumita Roy

Teaching Assistants: Indrajit Kalita, Veronica Naosekpam

Email-ids:

moumita@iiitg.ac.in

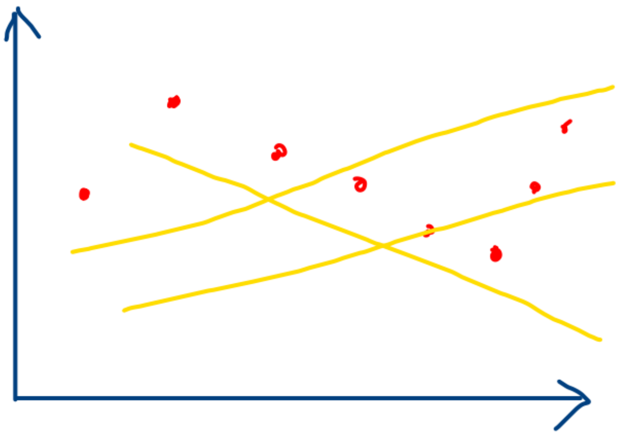
veronica.naosekpam@iiitg.ac.in
indrajit.kalita@iiitg.ac.in

Mobile No: +91-8420489325 (only for emergency quires)

Reference: https://www.cse.iitm.ac.in/~miteshk/CS6910/Slides/Lecture2.pdf

Regression vs. Classification (considering linear hypothesis)

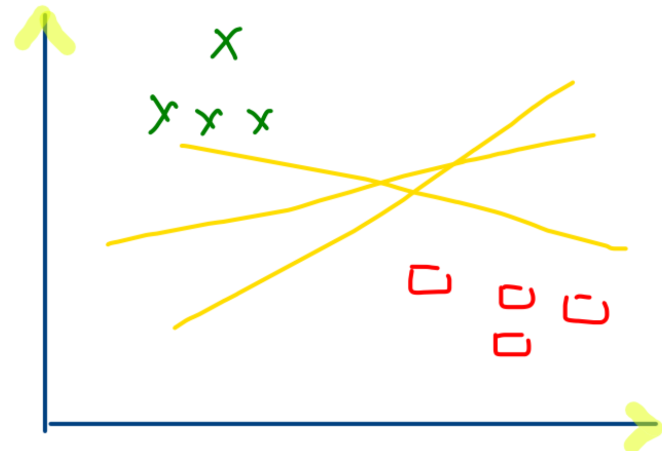
Regression



Linear Relation between independent and dependent variables

Try to find best fit one by minimizing cost function

classification

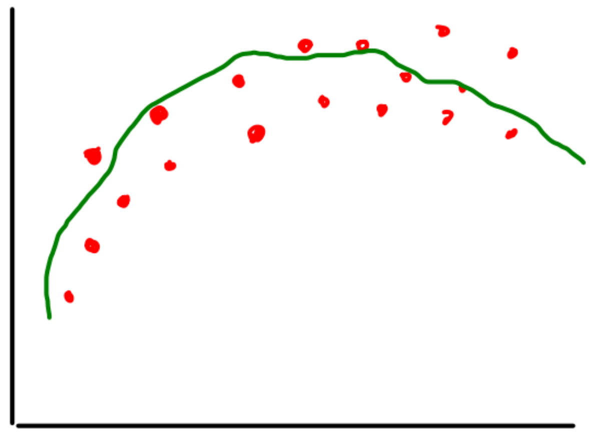


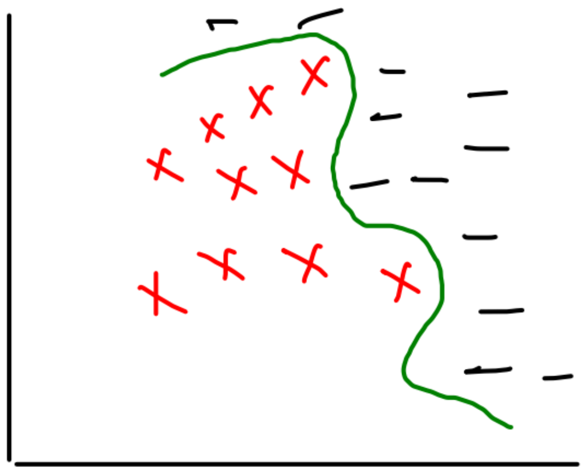
Data set is linearly separable

Try to find linear DB by minimizing cost function

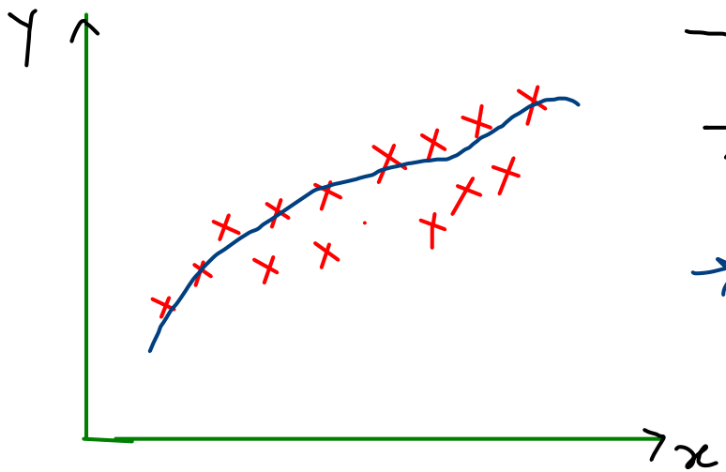
Consider binary classification

Scenarios (Non-linear hypothesis)





Polynomial Regression



$\rightarrow \boxed{h(x) = w_0 + w_1 x}$

$\rightarrow h(x) = w_0 + w_1 x + w_2 x^2$

$\rightarrow h(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

If, we consider, $h(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

Can we use linear regression to Solve the problem?

Introduce more features

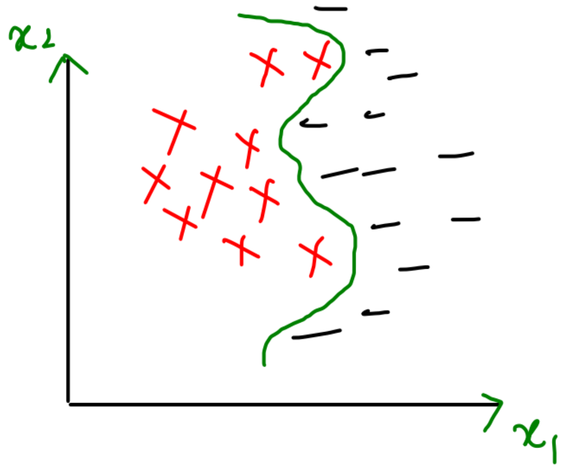
$$x_1 = x$$
$$x_2 = x^2$$
$$x_3 = x^3$$

Rewrite the hypothesis,

$$h(x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

Can we solve now?!

Non-linear classification



$\rightarrow h(x) = g(w_0 + w_1 x_1 + w_2 x_2)$

$\rightarrow h(x) = g(w_0 + w_1 x_1^2 + w_2 x_2^2$
$+ w_1 x_1 x_2 +$
$w_2 x_1 x_2^2 + \cdots)$

Introduce more features

$x_1' = x_1^2$

$x_2' = x_2^2$

$x_3' = x_1 x_2$

$\vdots$

so many....

$h(x) = g(w_0 + w_1 x_1' + w_2 x_2'$
$+ w_3 x_3' + \cdots)$

Can we solve this problem??

Observations

- Introducing more features (mainly polynomial terms; complex non-linear hypothesis)

- These features help the model to use existing learning algorithm to handle the problem (linear regression/logistic regression)

- Such hand-crafted feature engineering is not desirable option to solve non-linear problems (with more features)

ANN

- ANN is a very powerful and widely used model to learn a complex non-linear hypothesis
- ANN Representation helps us to automatically extract such features
- Solve XOR problem with ANN topics discussed so far (MP neuron, Perceptron, Sigmoid neuron)

XOR Problem (using MP neuron)

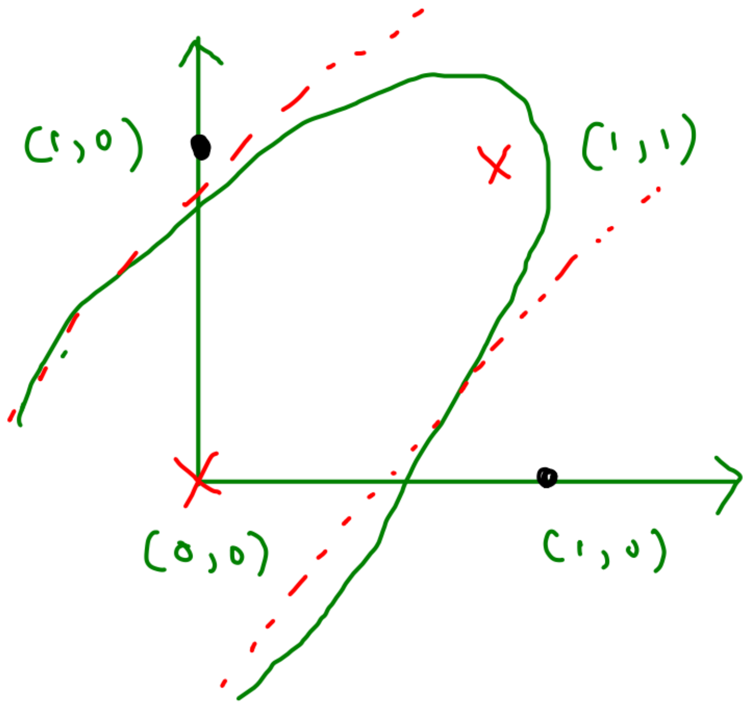
$y = 1, \quad \sum x_i \geq \theta$

$y = 0, \quad \sum x_i < \theta$

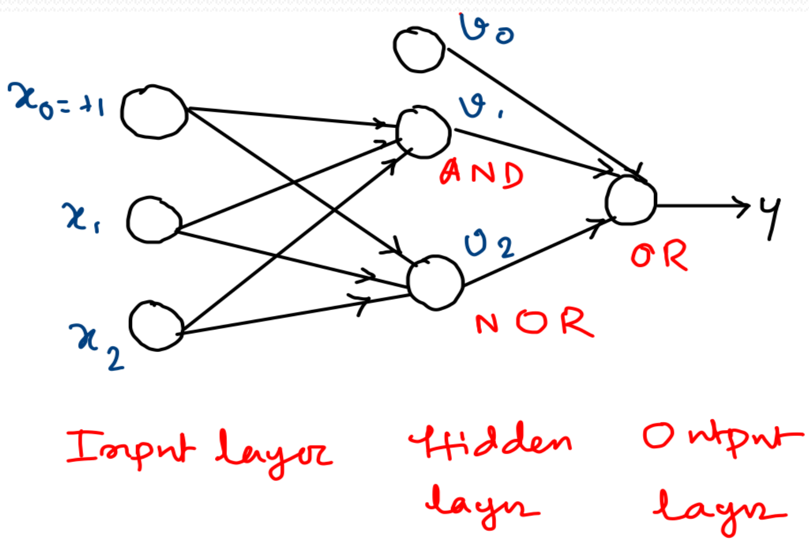
x_1	x_2	y	
0	0	1	$\rightarrow 0 \geq \theta$
0	1	0	$\rightarrow 1 < \theta$
1	0	0	$\rightarrow 1 < \theta$
1	1	1	$\rightarrow 2 \geq \theta$

Can we find θ !!

Geometrical representation for XOP Problem



Introduce multiple layers in ANN



x_0	x_1	v_1	v_2	y
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Hidden features (not always easy to demonstrate)

XOR → Simple problem (Solve by two lines)
we can solve it by dividing it into two linear problem (AND and NOR ; then OR)

Multilayer Perceptron (MLP)

- It has one input layer, one output layer, and multiple hidden layers (feed-forward architecture).
- Number of input neurons= number of features+1
- Number of output neurons=number of classes
- Number of hidden layers and hidden neurons in each layer are fixed experimentally (application depended)
- We consider, number of hidden layers=1
- Activation function has been considered in each hidden neuron and output neuron.

Neural representation for non-linear hypothesis

- MLPs can be trained to implement any given nonlinear input-output mapping (hypothesis).

- Here, multiple hidden neurons (with non-linear activation function) are helpful to extract hidden features automatically.

- MLP is basically transformation of the original input space into a new one, where the classification task becomes linearly separable.

- In this regard, Cover's theorem may be referred:

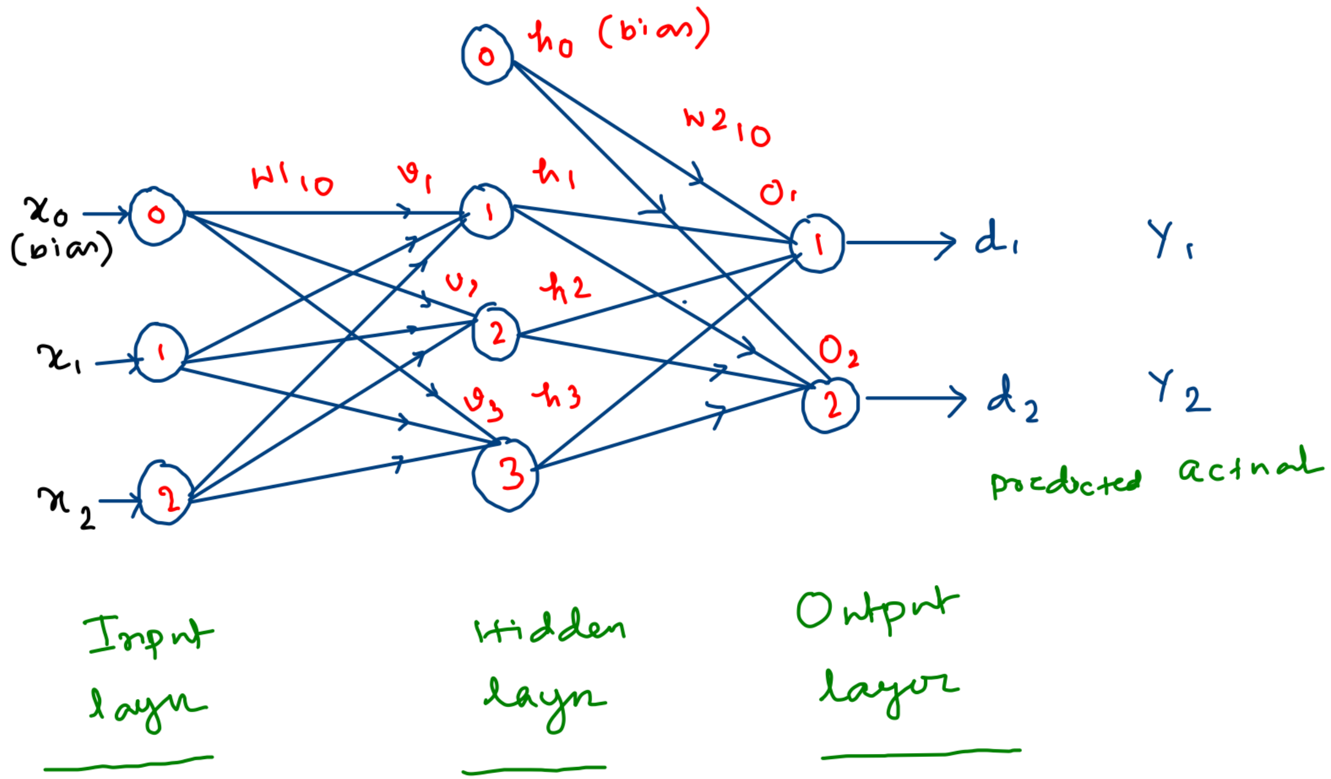
  A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space.

Training Process of MLP

- Design a MLP to solve a non-linearly separable problem (2 features and two classes)

  - Data preprocessing (one hot encoding)
  - Architecture
  - Training algorithm (?)
  - Cost function: MSE
  - Activation function: Sigmoid/logistic function

Architecture



Start training (using incremental GD)

Input to hidden layer

Random initialized the weights

$$v_1 = w1_{10} x_0 + w1_{11} x_1 + w2_{12} x_2$$

$v_2 =$ same ...

$v_3 =$ same ...

$$h_1 = g(v_1) = \frac{1}{1+\exp(-v_1)}$$

$h_2 =$ same ...

$h_3 =$ same ....

Hidden to output

$$O_1 = W2_{10} h_0 + W2_{11} h_1 + W2_{12} h_2 + W2_{13} h_3$$

$O_2 =$ Same ....

$$d_1 = g(O_1)$$

$$d_2 = g(O_2)$$

Training GD

$$E_{itr} = \frac{1}{2}\sum_{j=1}^{2} (y_j - d_j)^2$$

error in each output neuron.

$$e_1 = y_1 - d_1$$

$$e_2 = y_2 - d_2$$

$$w_{ji} := w_{ji} + \alpha (y_j - d_j) d_j (1 - d_j) x_i$$

Update the weight between hidden to output:

$$W2_{10} = W2_{10} + \alpha (y_1 - d_1) d_1 (1 - d_1) h_0$$

$$\vdots$$

Same for others

Update weight between input to hidden

$$w'_{10} = w_{10} + \alpha \underbrace{(? - h_1)}_{?} h_1 (1 - h_1) x_0$$

error (not known)

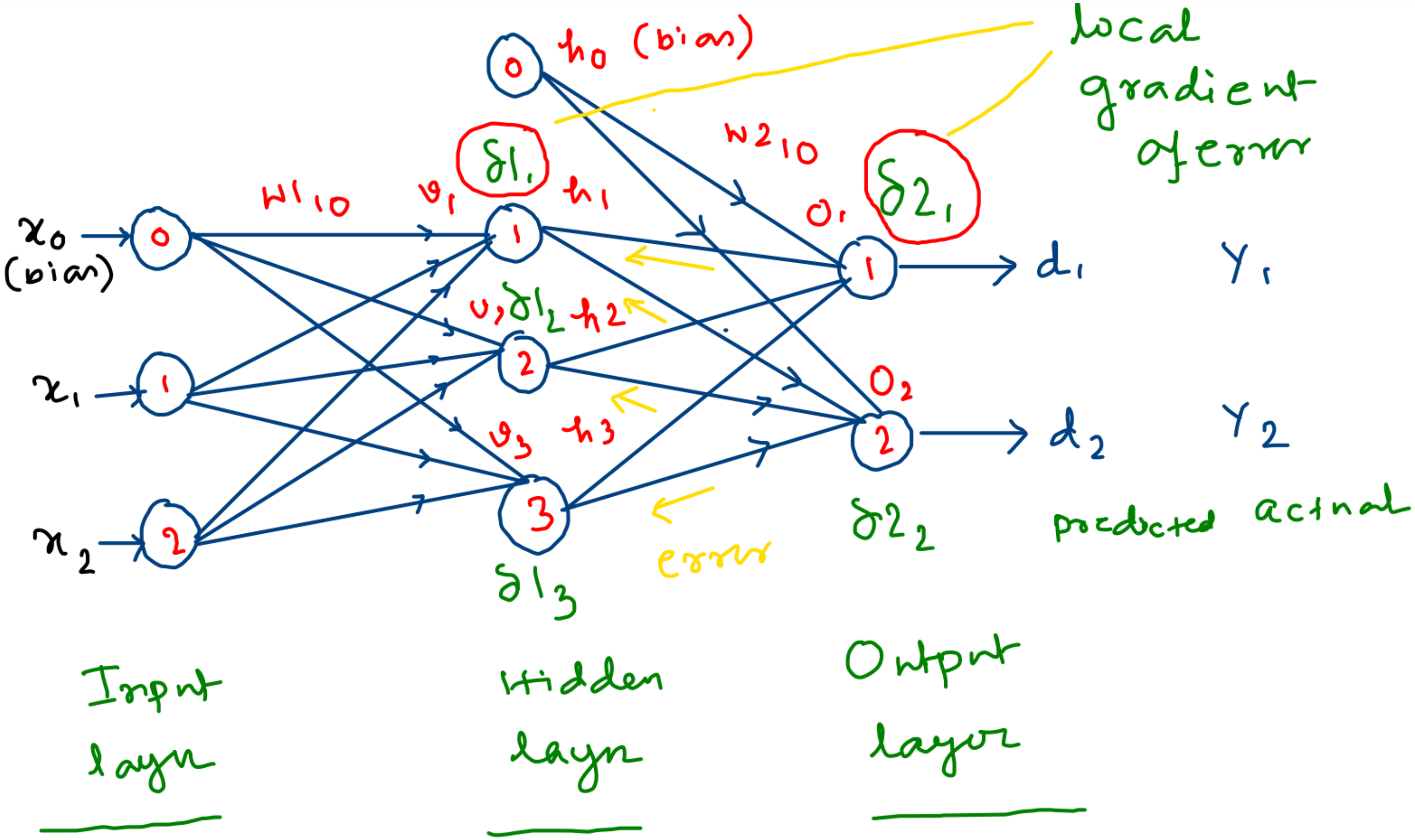
Solution: back-propagating errors for output neurons to hidden neurons

Back-propagation learning algorithm

Step 1: Forward pass

Step 2: Backward pass

Back-propagation algorithm



Back-propagation algorithm

Forward Pass (updation for one sample)

Input to hidden layer

$$v_1 = w1_{10} x_0 + w1_{11} x_1 + w2_{12} x_2$$

$v_2 =$ same ...

$v_3 =$ same ...

$$h_1 = g(v_1) = \frac{1}{1+\exp(-v_1)}$$

$h_2 =$ same ...

$h_3 =$ same ...

Hidden to output

$$O_1 = W2_{10} h_0 + W2_{11} h_1 + W2_{12} h_2 + W2_{13} h_3$$

$O_2 =$ Same ....

$$d_1 = g(O_1)$$

$$d_2 = g(O_2)$$

Backward pass

Error calculation for the pattern

$$E_{1tr} = \frac{1}{2}\sum_{j=1}^{2}(d_j - y_j)^2$$

weight updation in GD

$$w_{ji} := w_{ji} + \alpha \underbrace{(y_j - d_j)(1-d_j)\,d_j}\, x_i$$

error in each output node

$$e_1 = (y_1 - d_1)$$

$$e_2 = (y_2 - d_2)$$

local gradient of error in each output node

$$\delta 2_1 = d_1 (1 - d_1)(y_1 - d_1)$$

$$\delta 2_2 = d_2 (1 - d_2)(y_2 - d_2)$$

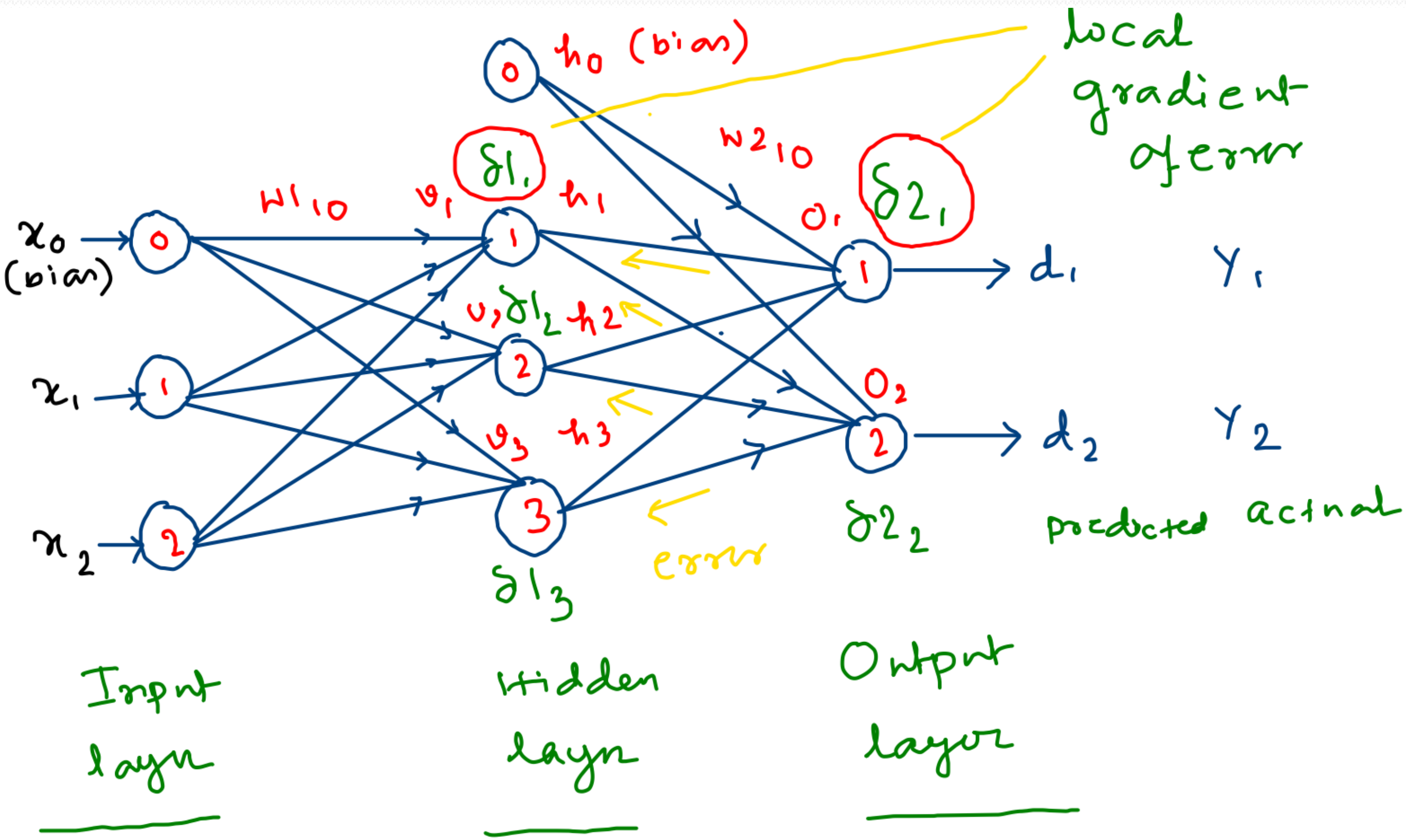
local gradient of error in each hidden node

$$\delta 1_1 = (\delta 2_1 * W2_{11} + \delta 2_2 * W2_{21})\, h_1 (1 - h_1)$$

$$\delta 1_2 = (\delta 2_1 * W2_{12} + \delta 2_2 * W2_{22})\, h_2 (1 - h_2)$$

$$\delta 1_3 = \;?$$

Back-propagation algorithm



Updation of weights

$$w_{ji} := w_{ji} + \alpha (y_j - d_j) d_j (1 - d_j) x_i$$

hidden to output

$$w2_{10} := w2_{10} + \alpha * \delta 2_1 * h_0$$

$$w2_{11} := w2_{11} + \alpha * \delta 2_1 * h_1$$

$\vdots$

Input to hidden

$$W1_{10} := W1_{10} + \alpha * \delta 1_1 * x_0$$

$$W1_{11} := W1_{11} + \alpha * \delta 1_1 * x_1$$

$$\vdots$$

Stopping criteria

no. of samples

$$\text{epoch 1}: \quad E_{itr} = \sum_{i=1}^{m} \sum_{j=1}^{2} (y_j - d_j)^2$$

no. of classes

Update weights for each sample

$$\text{epoch 2}: \quad E_{itr+1} = \sum_{i=1}^{m} \sum_{j=1}^{2} (y_j - d_j)^2$$

Until

$$|E_{itr-1} - E_{itr}| > \text{small value or}$$

$$\text{no. of epochs} < \text{large value}$$