Foundations of Machine Learning

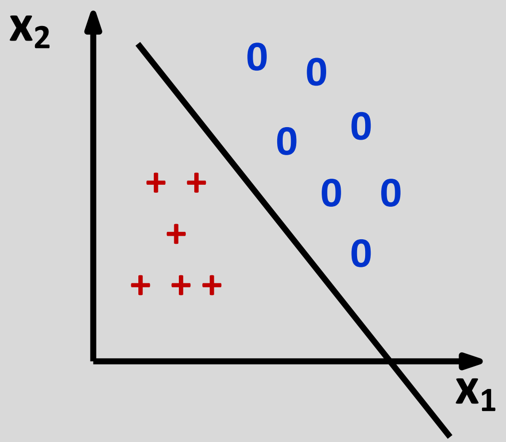
Module 6: Neural Network

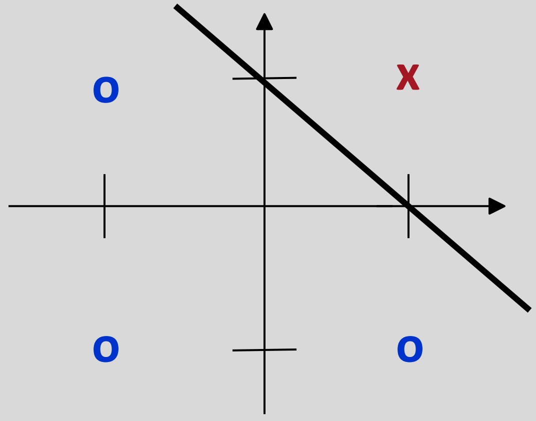
Part C: Neural Network and Backpropagation Algorithm

Sudeshna Sarkar
IIT Kharagpur

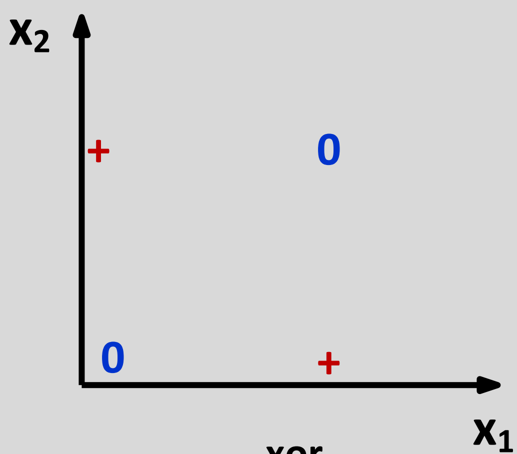
Single layer Perceptron

- Single layer perceptrons learn linear decision boundaries

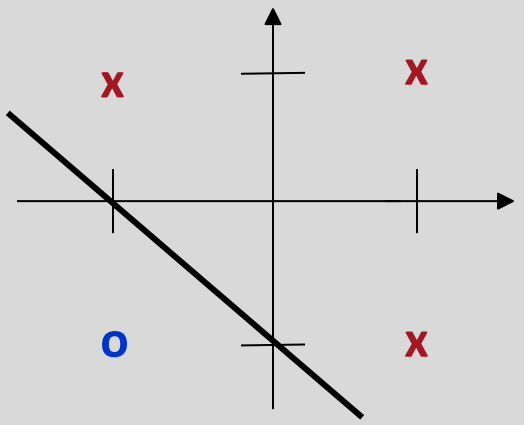




x: class I (y = 1)
o: class II (y = -1)



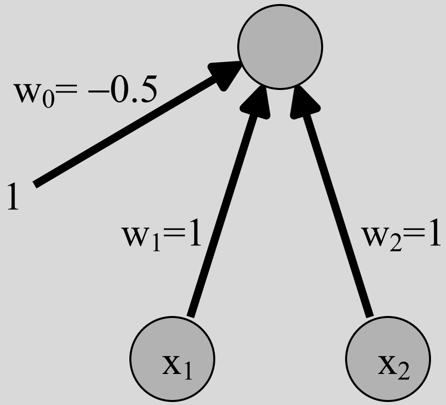
xor

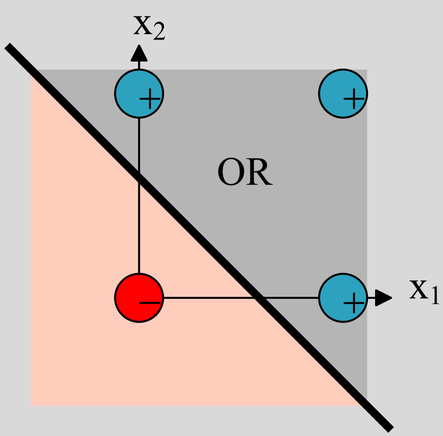


x: class I (y = 1)
o: class II (y = -1)

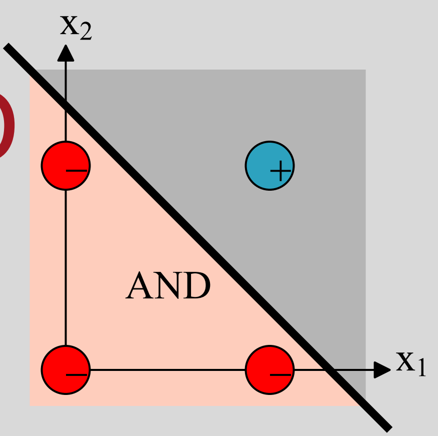
Boolean OR

input x1	input x2	ouput
0	0	0
0	1	1
1	0	1
1	1	1

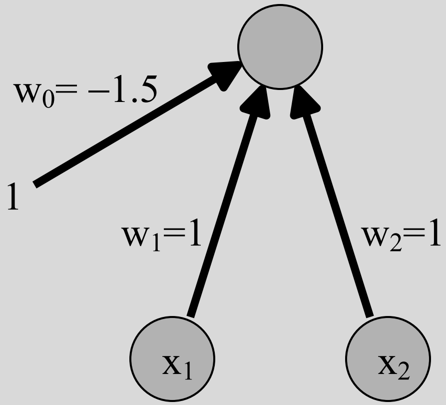




Boolean AND

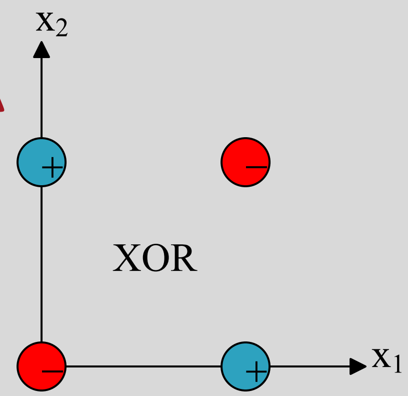


input x1	input x2	ouput
0	0	0
0	1	0
1	0	0
1	1	1



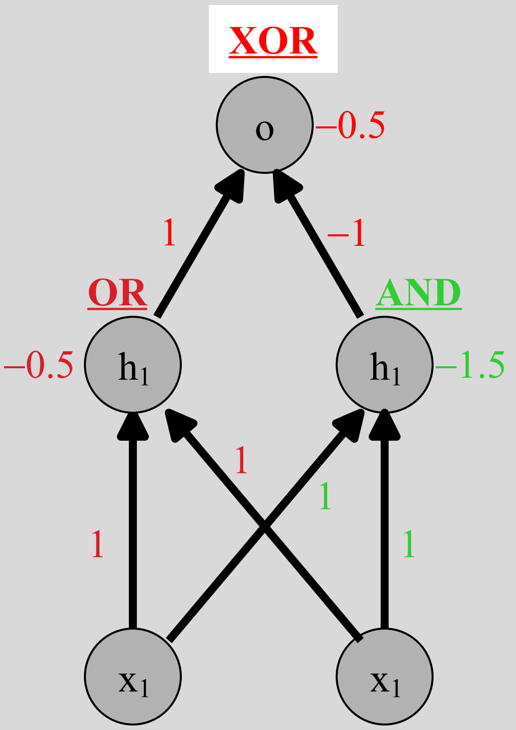
Boolean XOR

input x1	input x2	ouput
0	0	0
0	1	1
1	0	1
1	1	0



Boolean XOR

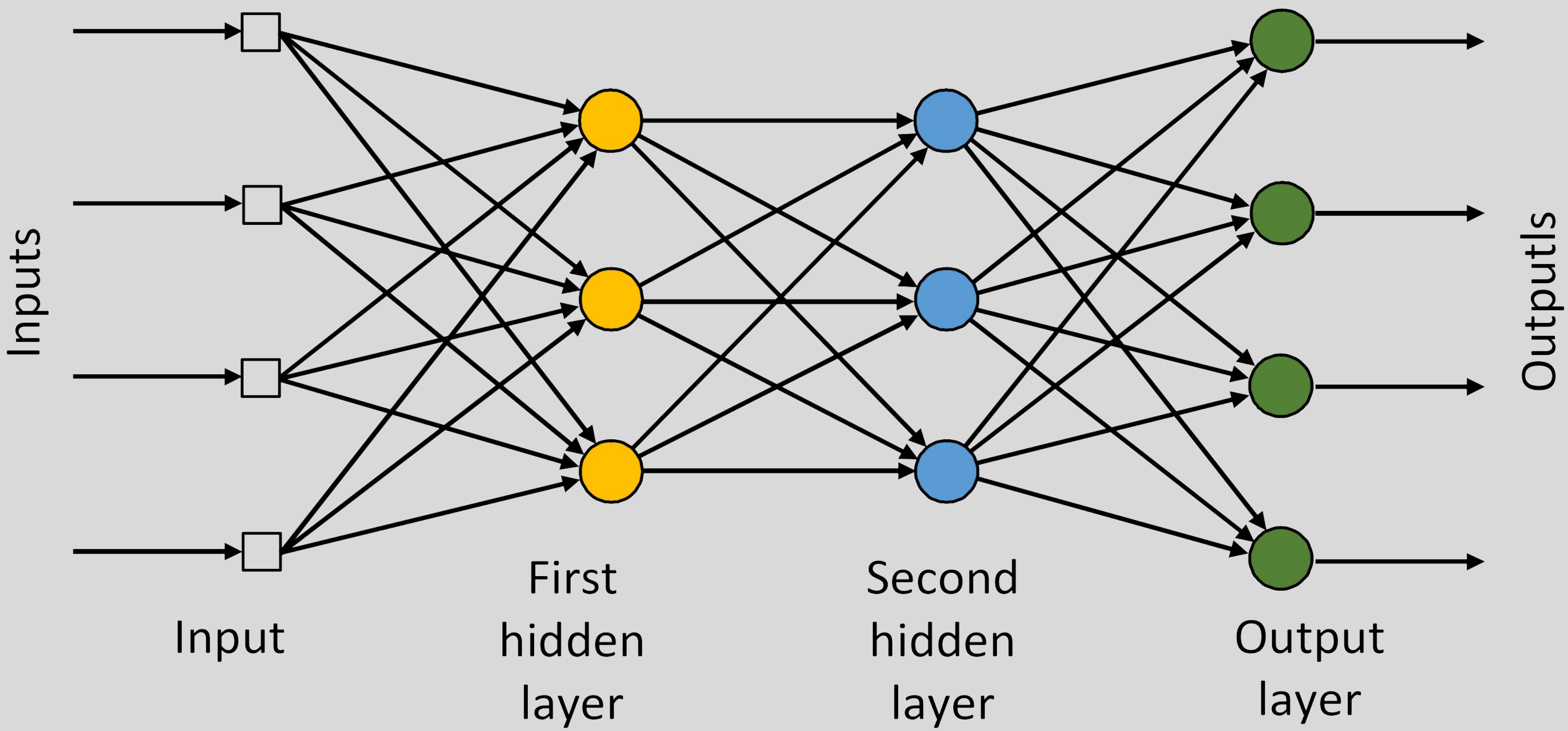
input x1	input x2	ouput
0	0	0
0	1	1
1	0	1
1	1	0



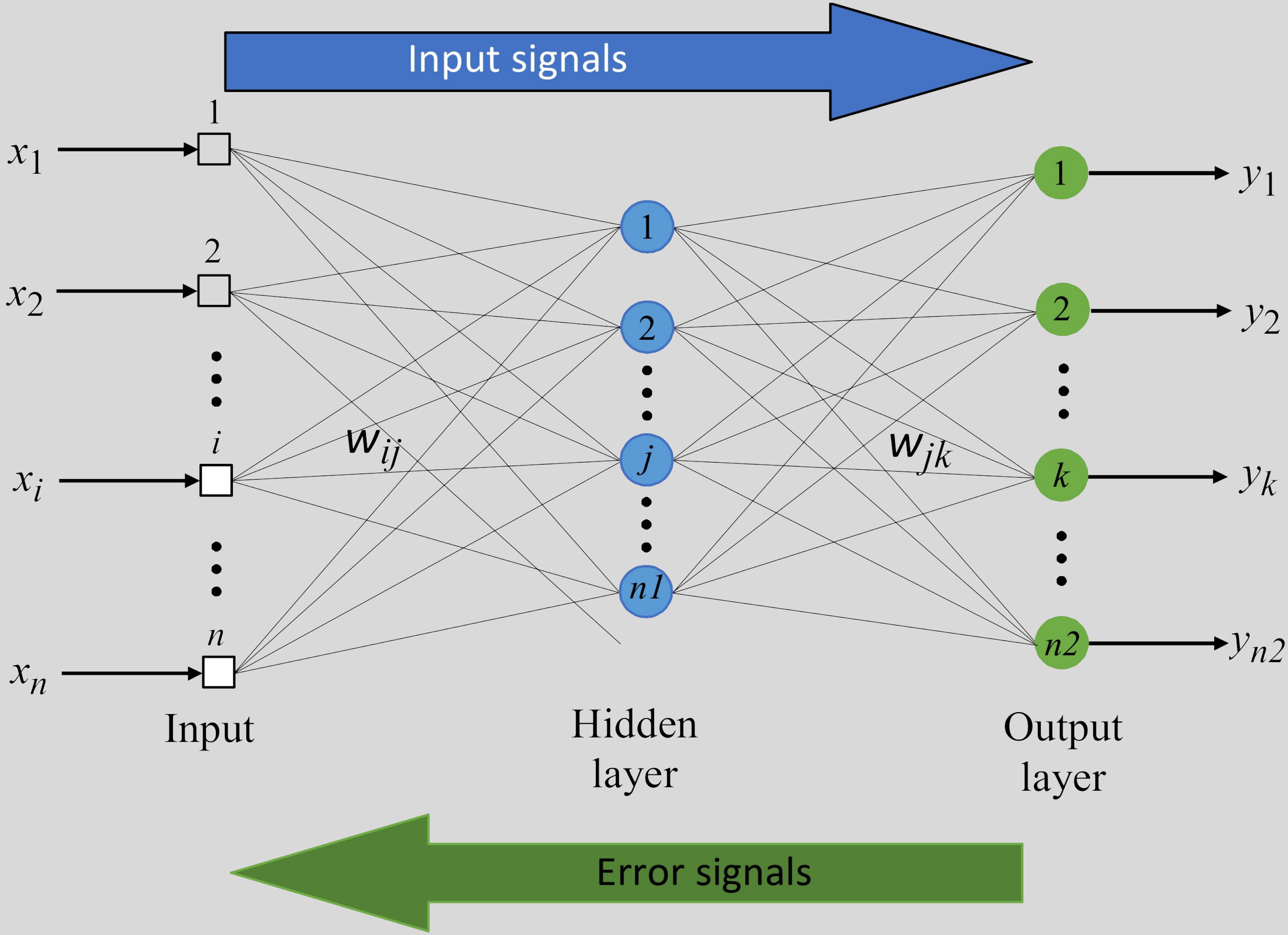
Representation Capability of NNs

- Single layer nets have limited representation power (linear separability problem). Multi-layer nets (or nets with non-linear hidden units) may overcome linear inseparability problem.
- Every Boolean function can be represented by a network with a single hidden layer.
- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers.

Multilayer Network



Two-layer back-propagation neural network



Derivation

- For one output neuron, the error function is

$E = \frac{1}{2}(y - o)^2$

- For each unit j, the output o_j is defined as

$o_j = \varphi\left(net_j\right) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$

The input net_j to a neuron is the weighted sum of outputs o_k of previous n neurons.

- Finding the derivative of the error:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

For one output neuron, the error function is $E = \frac{1}{2}(y - o)^2$

For each unit j, the output o_j is defined as

$$o_j = \varphi(net_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$= \sum_l \left(\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial net_{z_l}} w_{jz_l} \right) \varphi(net_j)\left(1 - \varphi(net_j)\right) o_i$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

with

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} = \left\{ \begin{array}{ll} (o_j - y_j)\, o_j (1 - o_j) & \text{if } j \text{ is an output neuron} \\ \left(\sum_Z \delta_{z_l} w_{jl}\right) o_j (1 - o_j) & \text{if } j \text{ is an inner neuron} \end{array} \right\}$$

To update the weight w_{ij} using gradient descent, one must choose a learning rate η.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Backpropagation Algorithm

Initialize all weights to small random numbers.
Until satisfied, do

- For each training example, do
  - Input the training example to the network and compute the network outputs
  - For each output unit k

    $\delta_k \leftarrow o_k(1-o_k)(y_k-o_k)$
  - For each hidden unit h

    $\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{h,k}, \delta_k,$
  - Update each network weight w_i, j

    $w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$

    where

    $\Delta w_{i,j} = \eta \delta_j x_{i,j}$

x_d = input

y_d = target output

o_d = observed unit output

w_{ij} = wt from i to j

Backpropagation

- Gradient descent over entire network weight vector
- Can be generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
- May include weight momentum α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

- Training may be slow.
- Using network after training is very fast

Training practices: batch vs. stochastic vs. mini-batch gradient descent

- **Batch gradient descent**:
  1. Calculate outputs for the entire dataset
  2. Accumulate the errors, back-propagate and update
- **Stochastic/online gradient descent**:
  1. Feed forward a training example
  2. Back-propagate the error and update the parameters
- **Mini-batch gradient descent**:

Too slow to converge
Gets stuck in local minima

Converges to the solution faster
Often helps get the system out of local minima

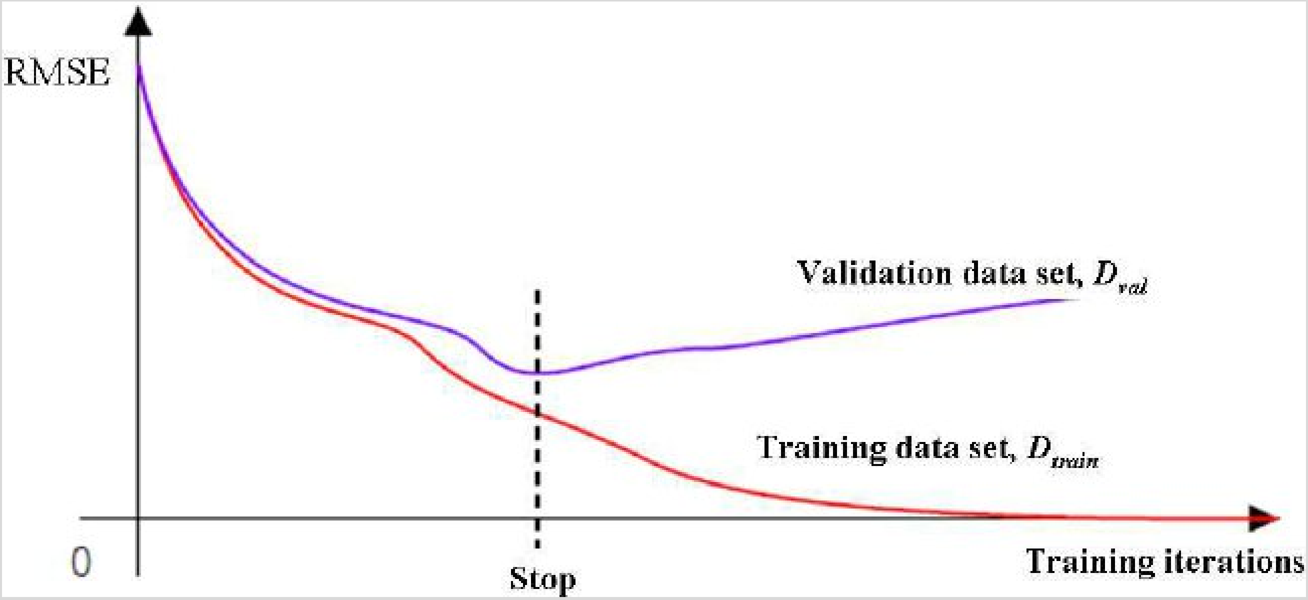
Learning in *epochs* Stopping

- Train the NN on the entire training set over and over again
- Each such episode of training is called an "epoch"

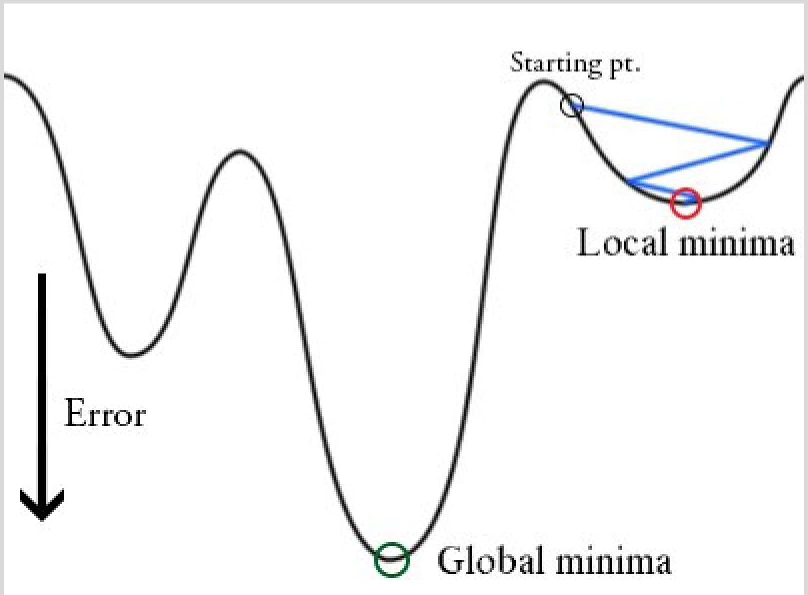
Stopping

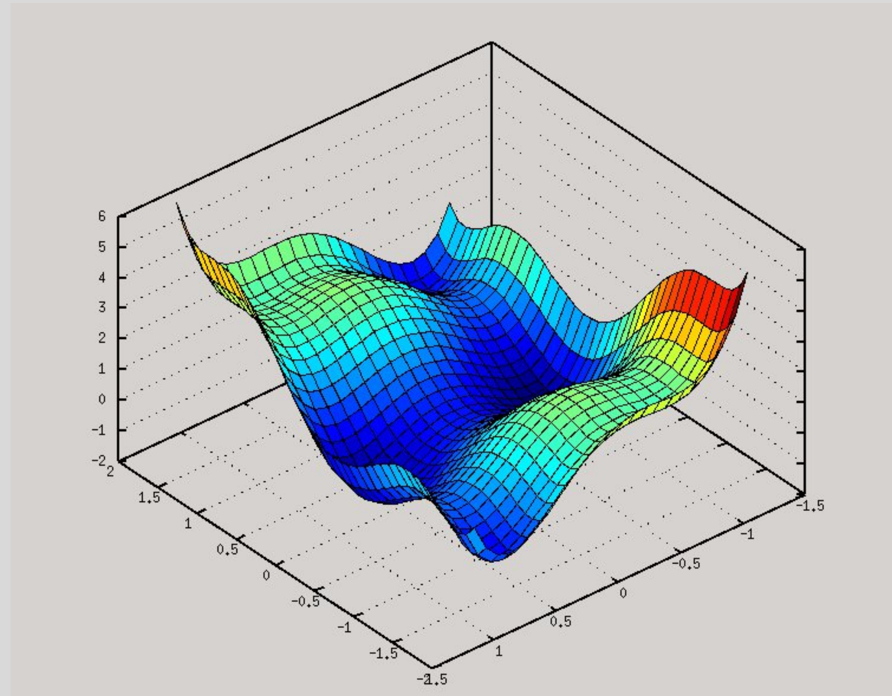
1. Fixed maximum number of epochs: most naïve
2. Keep track of the training and validation error curves.

Overfitting in ANNs



Local Minima





- NN can get stuck in local minima for small networks.
- For most large networks (many weights) local minima rarely occurs.
- It is unlikely that you are in a minima in every dimension simultaneously.

ANN

- Highly expressive non-linear functions
- Highly parallel network of logistic function units
- Minimizes sum of squared training errors
- Can add a regularization term (weight squared)
- Local minima
- Overfitting

Thank You