UiO : Department of Informatics
University of Oslo

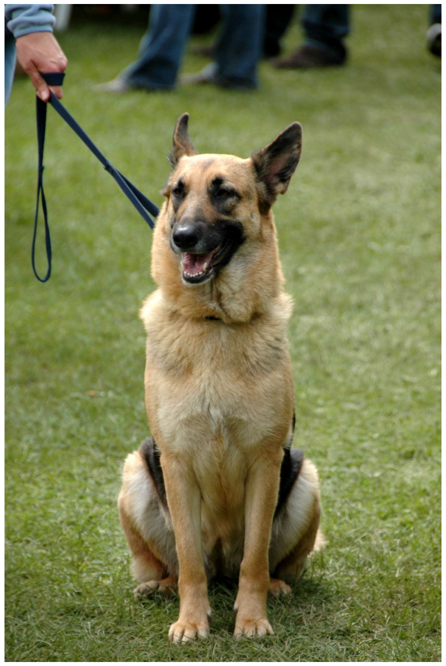
INF3490 - Biologically inspired computing

Lecture 28th September 2016

Multi-Layer Neural Networks

Kai Olav Ellefsen

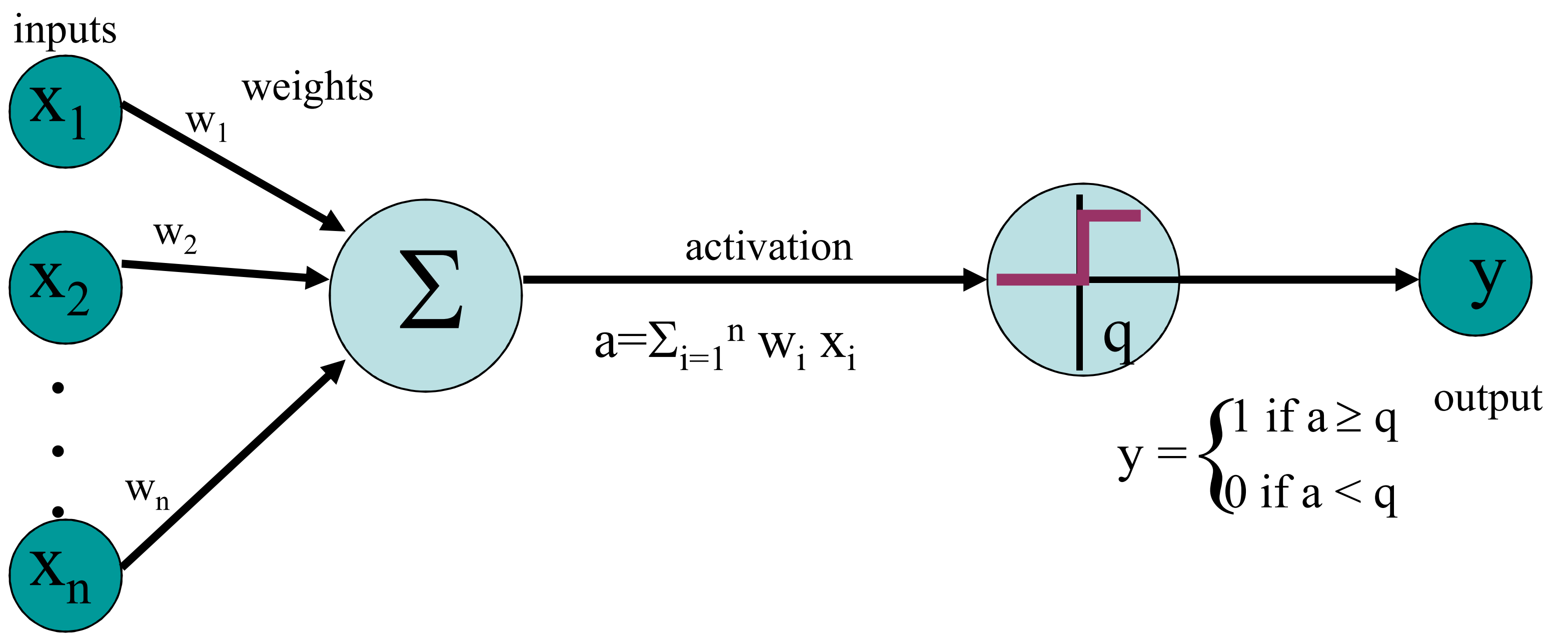
Classification



Trained Classifier

“DOG”

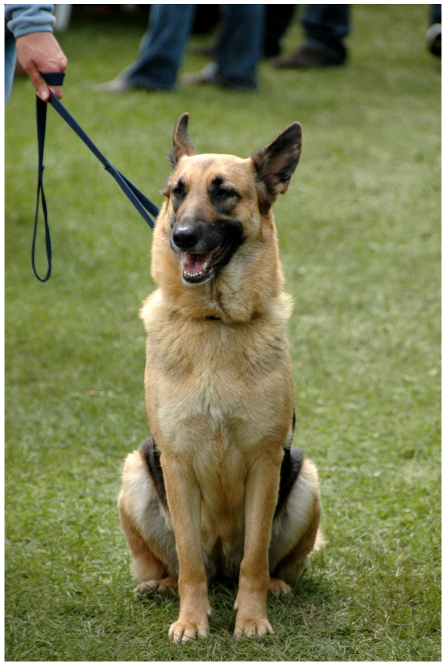
Perceptron

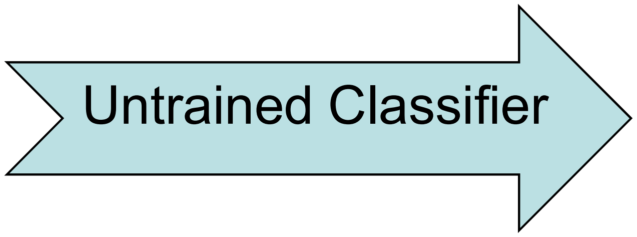


$a=\Sigma_{i=1}^{n} w_i x_i$

$$y = \begin{cases} 1 \text{ if } a \geq q \\ 0 \text{ if } a < q \end{cases}$$

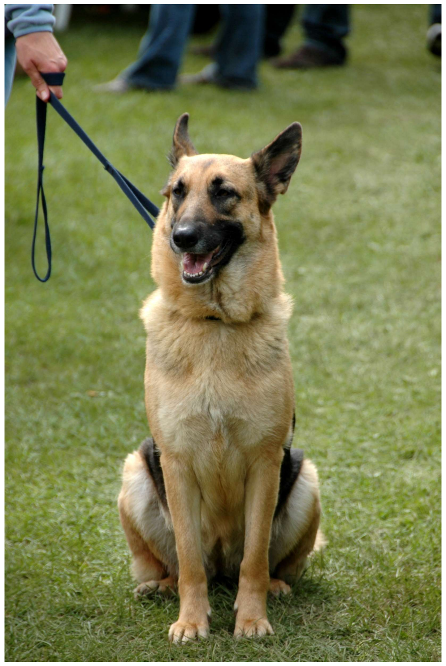
Training a classifier (supervised learning)

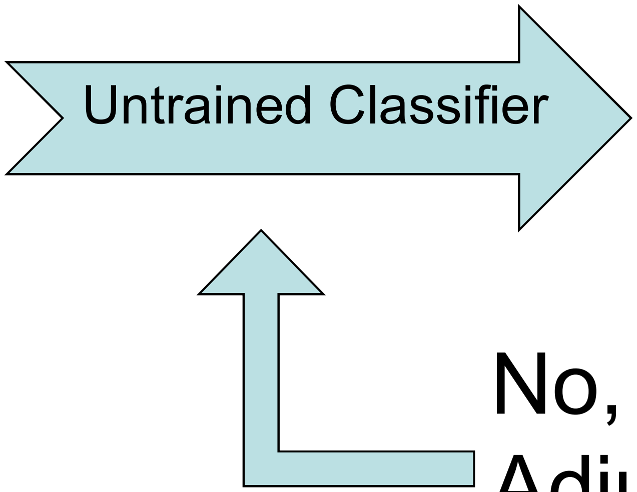




“CAT”

Training a classifier (supervised learning)

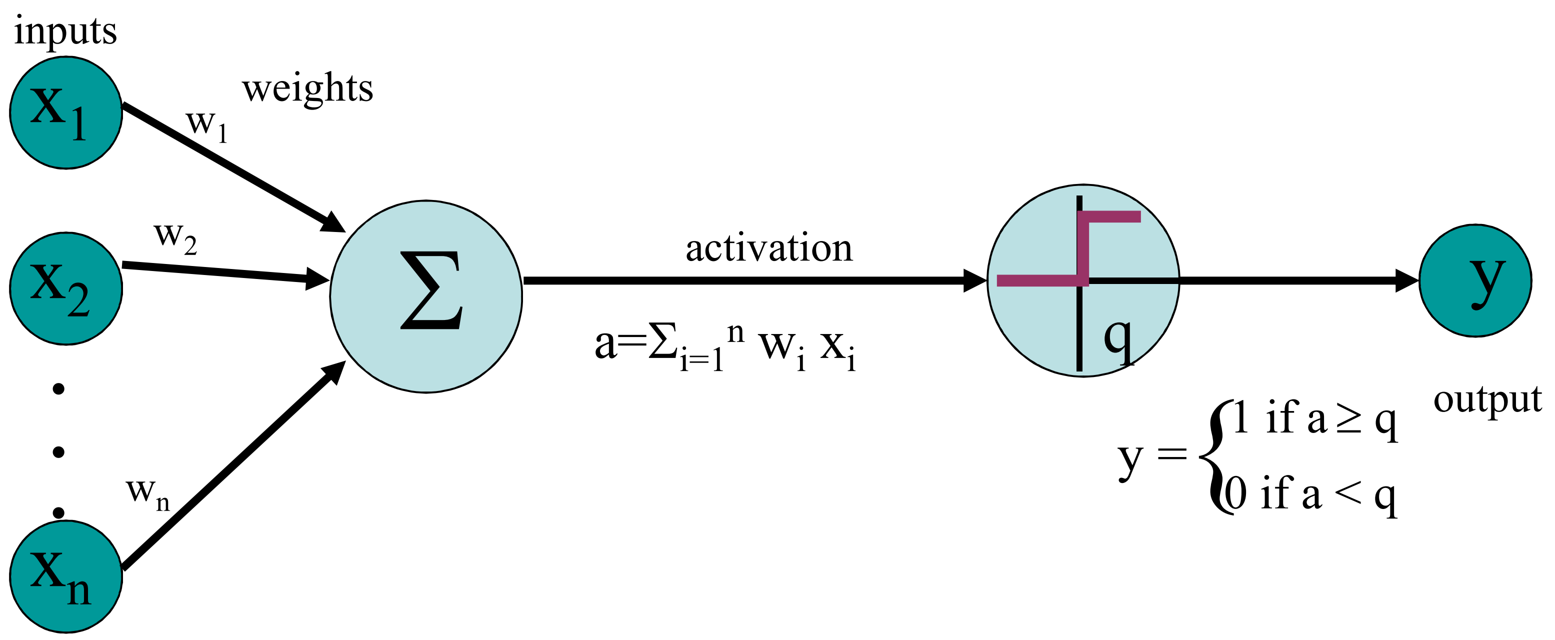




“CAT”

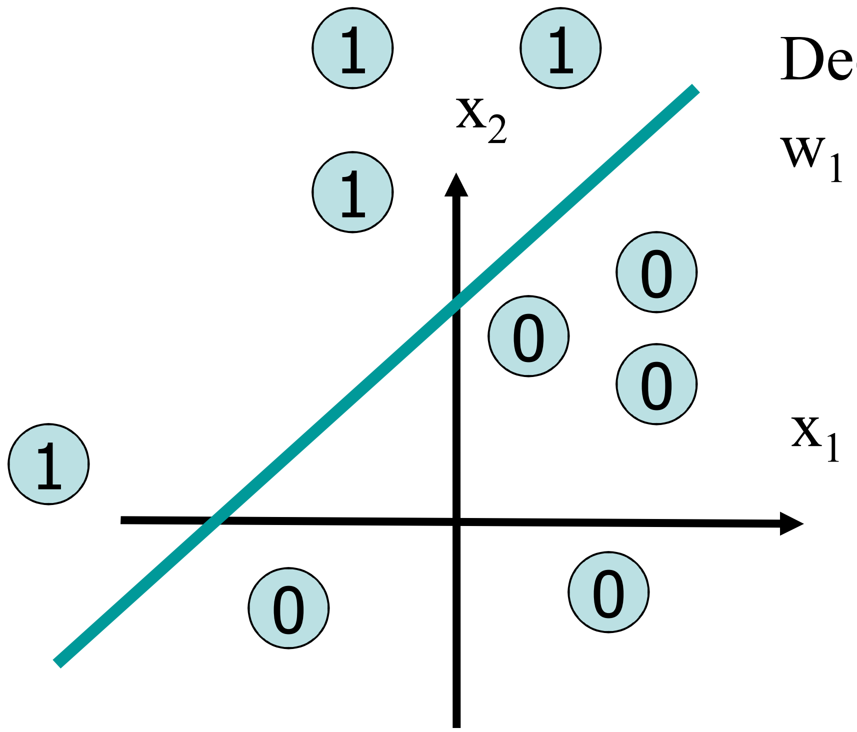
No, it was a dog.
Adjust classifier parameters

Training a perceptron


inputs
x_1
x_2
x_n
weights
w_1
w_2
w_n
Σ
activation
$a=\Sigma_{i=1}^{n} w_i x_i$
q
y
output
$y = \begin{cases} 1 \text{ if } a \geq q \\ 0 \text{ if } a < q \end{cases}$

$$\Delta w_{ij} = \overset{\text{Learning rate}}{\eta} \cdot \underbrace{(\underset{\text{Desired output}}{t_j} - \underset{\text{Actual output}}{y_j})}_{\text{Error}} \cdot \overset{\text{Input}}{x_i}$$

Decision Surface



Decision line

$w_1\,x_1 + w_2\,x_2 = q$

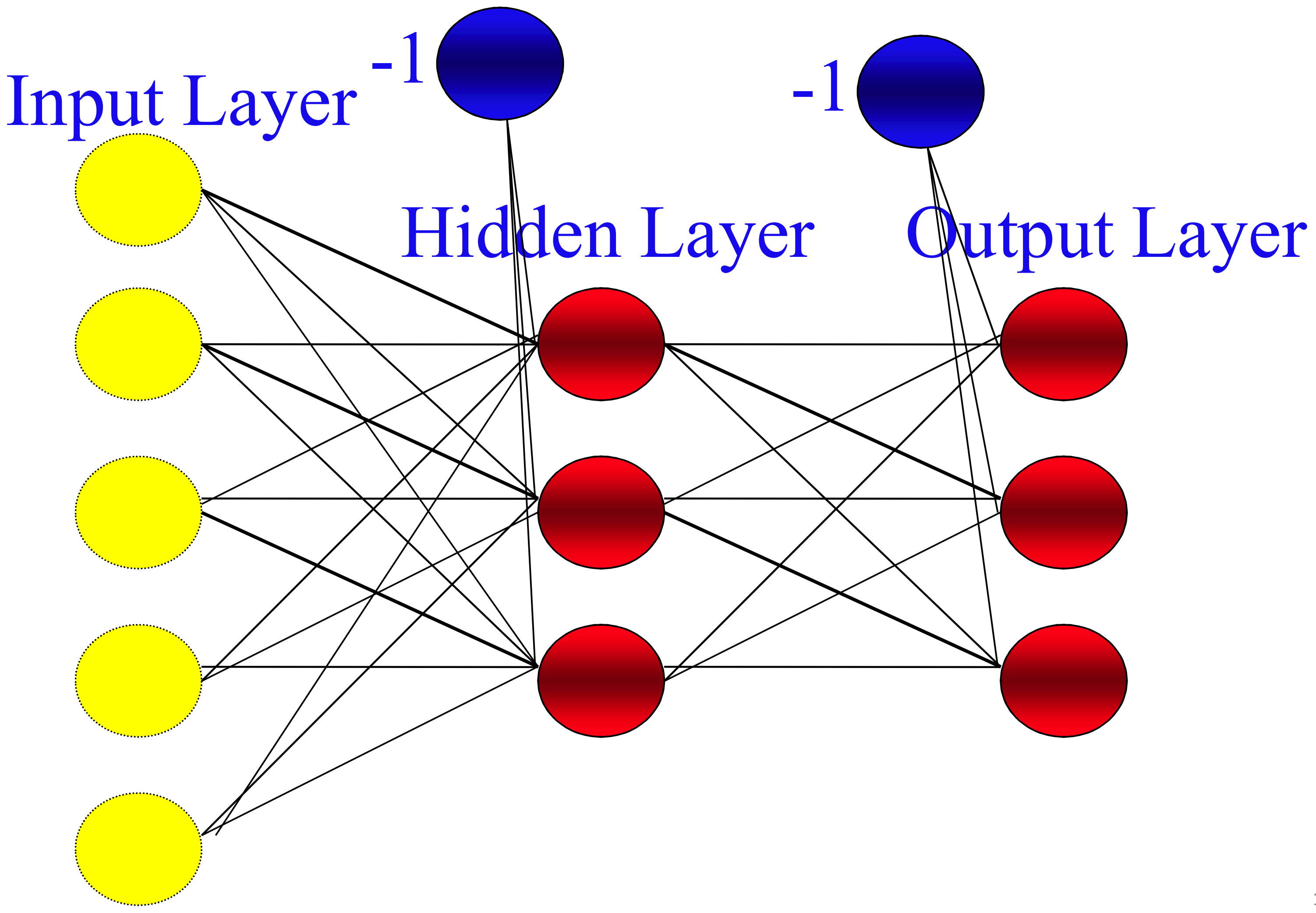
A Quick Overview

- Linear Models are easy to understand.
- However, they are very simple.
  - They can only identify flat decision boundaries (straight lines, planes, hyperplanes, ...).
- ***Majority of interesting data are not linearly separable. Then?***

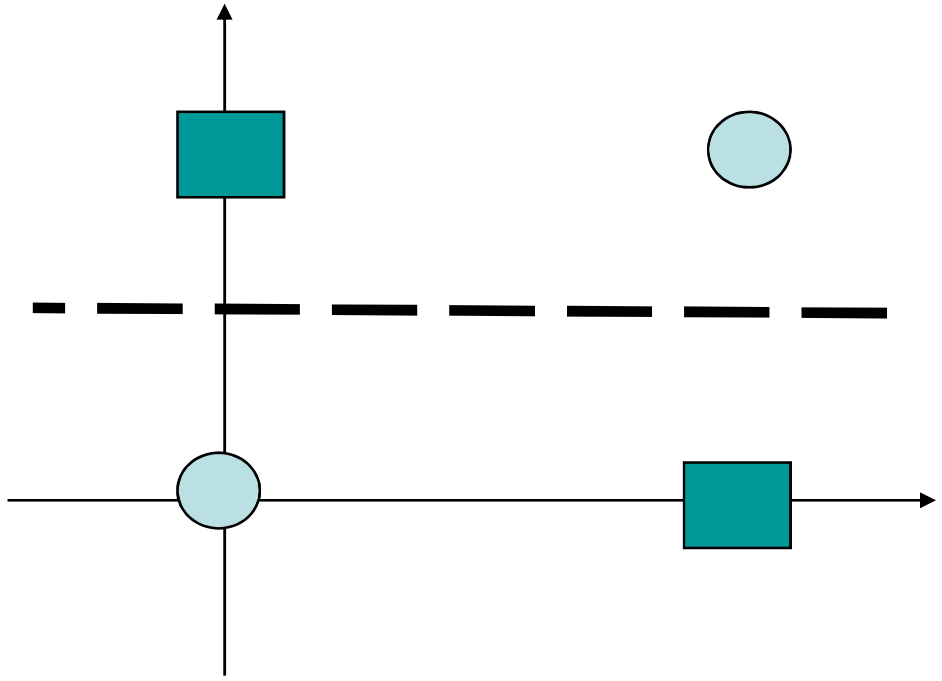
A Quick Overview

- ***Learning*** in the neural networks (NN) happens in the weights.
- **Weights** are associated with connections.
- Thus, it is sensible to add more connections to perform more complex computations.
- Two ways for non-lin. separation (not exclusive):
  - ***Recurrent Network***: connect the output neurons to the inputs with feedback connections.
  - ***Multi-layer perceptron network***: add neurons between the input nodes and the outputs.

Multi-Layer Perceptron (MLP)



XOR Problem



XOR (Exclusive OR) Problem

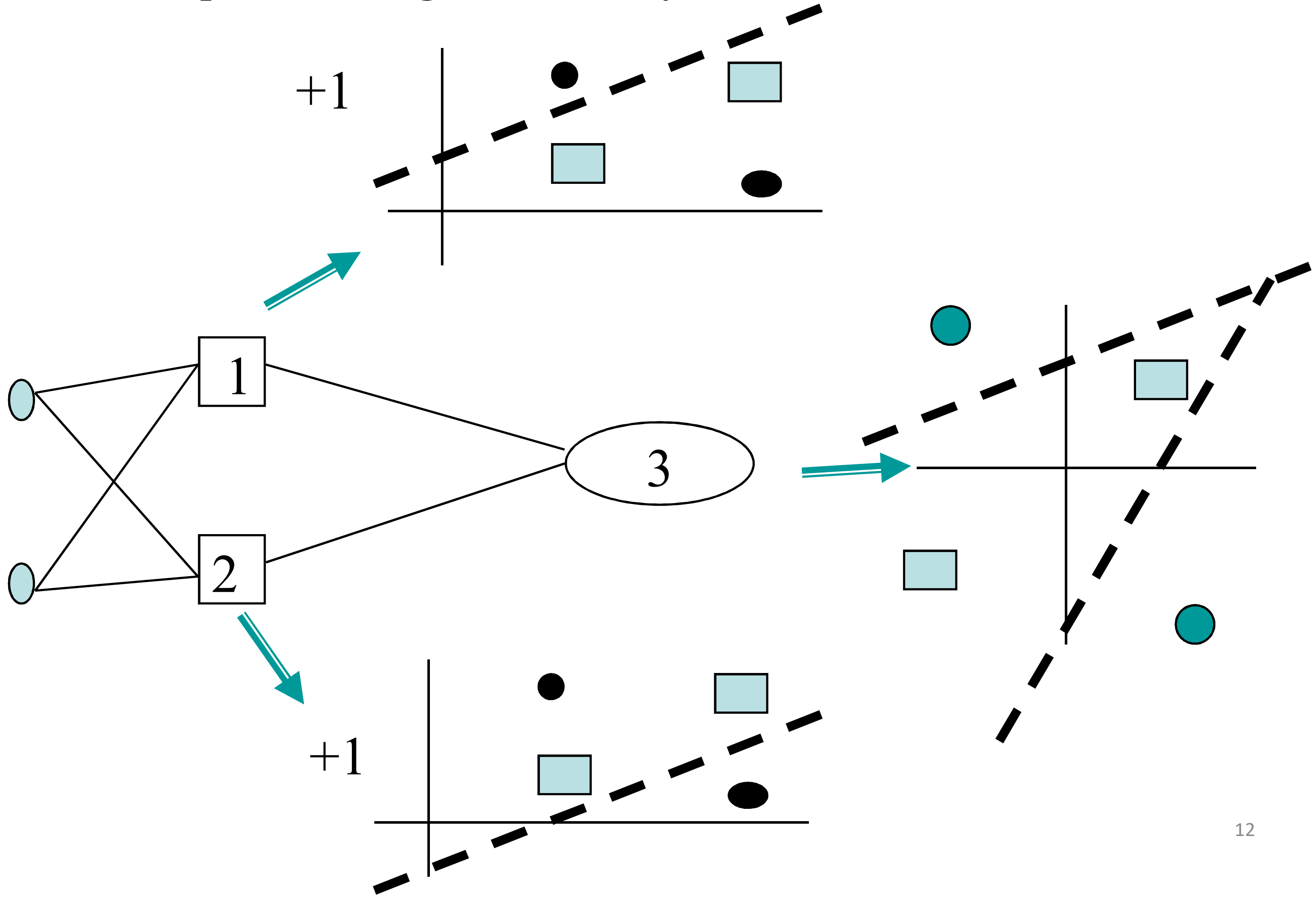
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Perceptron does not work here.

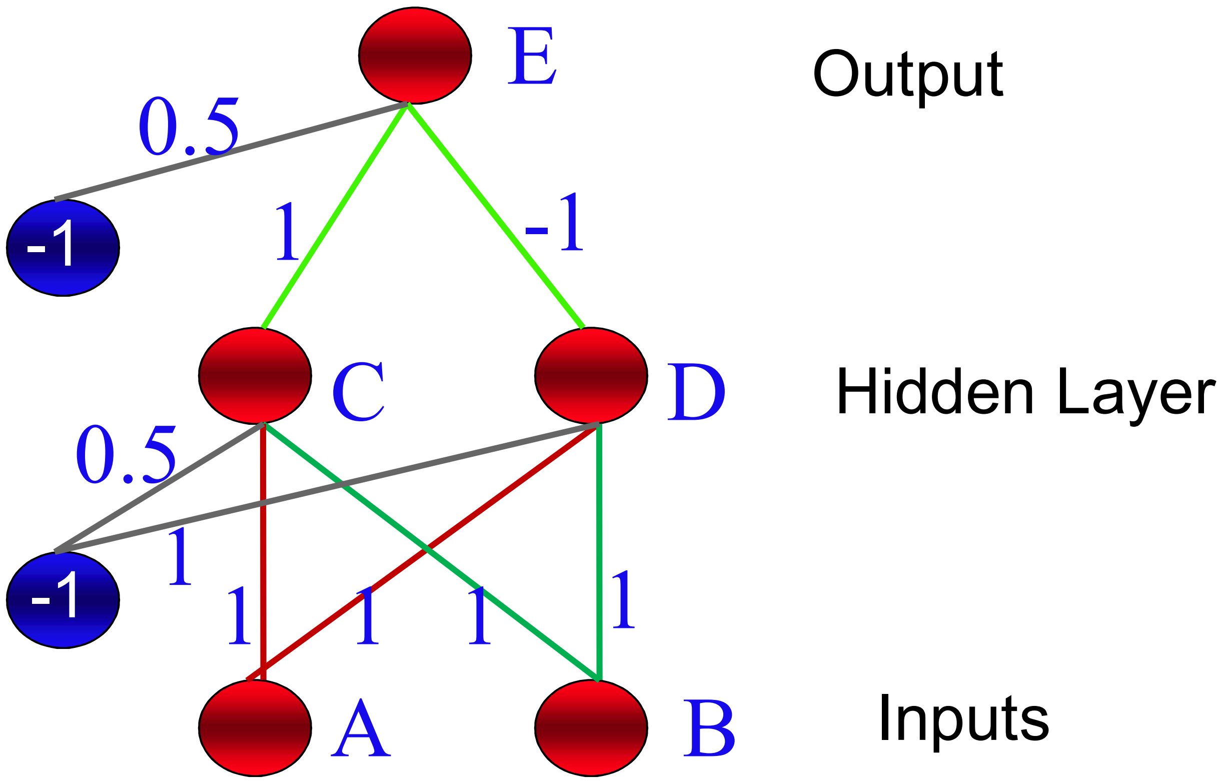
Single layer generates a linear decision boundary.

Solution for XOR : Add a Hidden Layer !!

Minsky & Papert (1969) offered **solution to XOR problem** by combining perceptron unit responses **using a second layer of units**.

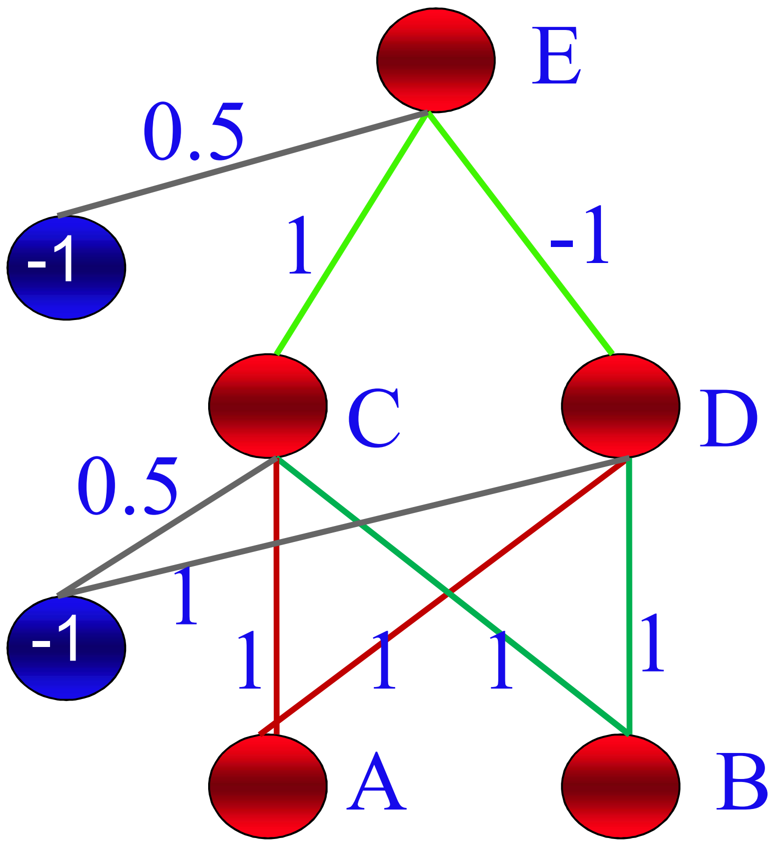


XOR Again



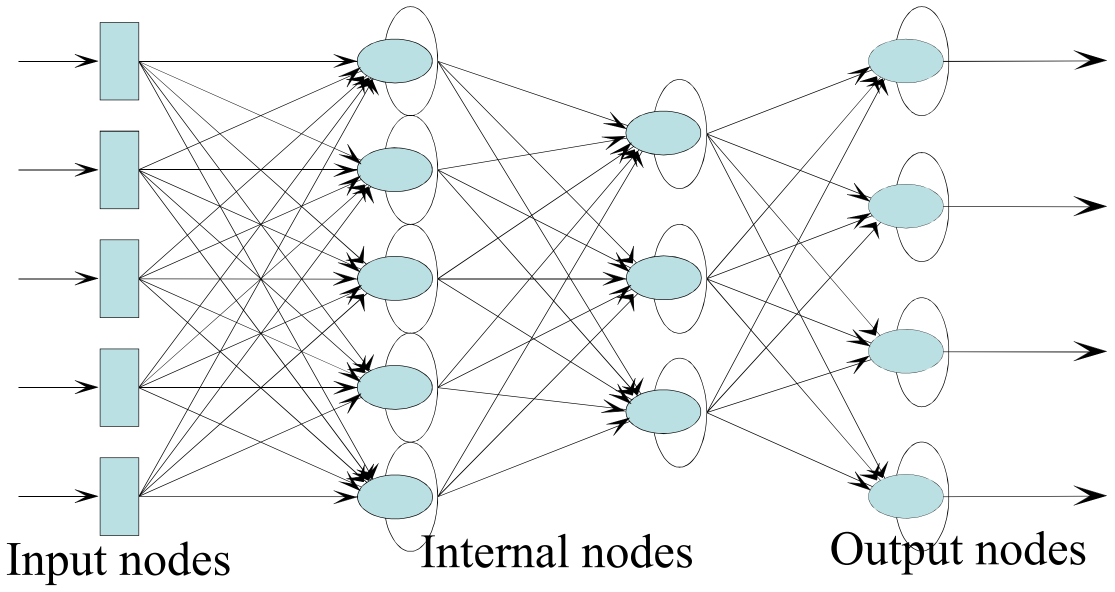
XOR Again

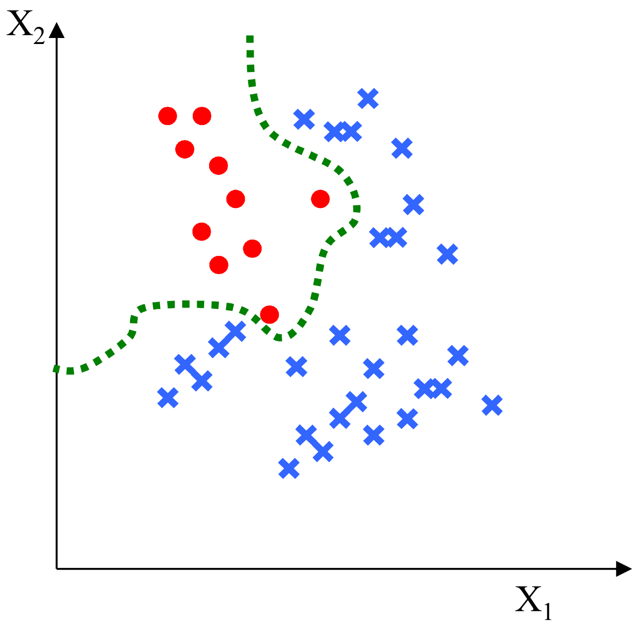
A	B	C_{in}	C_{out}	D_{in}	D_{out}	E_{in}
0	0	-0.5	0	-1	0	-0.5
0	1	0.5	1	0	0	0.5
1	0	0.5	1	0	0	0.5
1	1	1.5	1	1	1	-0.5



MLP Decision Boundary – Nonlinear Problems, Solved!

In contrast to perceptrons, multilayer networks can learn not only multiple decision boundaries, but the boundaries may also be nonlinear.





Multilayer Network Structure

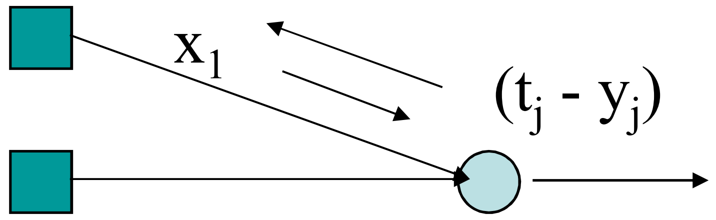
- A neural network with one or ***more*** layers of ***nodes*** between the input and the output nodes is called ***multilayer network***.
- The multilayer *network structure*, or *architecture*, or *topology*, consists of ***an input layer*, *one or more hidden layers***, and ***one output layer***.
- The input nodes pass values to the first hidden layer, its nodes to the second and so until producing outputs.
    - A network with a layer of input units, a layer of hidden units and a layer of output units is a ***two-layer network***.
    - A network with two layers of hidden units is a ***three-layer network***, and so on.

Properties of the Multi-Layer Perceptron

- No connections within a single layer.
- No direct connections between input and output layers.
- Fully connected; all nodes in one layer connect to all nodes in the next layer.
- Number of output units need not equal number of input units.
- Number of hidden units per layer can be more or less than input or output units.

How to Train MLP?

- How we can train the network, so that
  - The weights are adapted to generate correct (target answer)?



- In Perceptron, errors are computed at the output.
- In MLP,
  - Don't know which weights are wrong:
  - Don't know the correct activations for the neurons in the hidden layers.

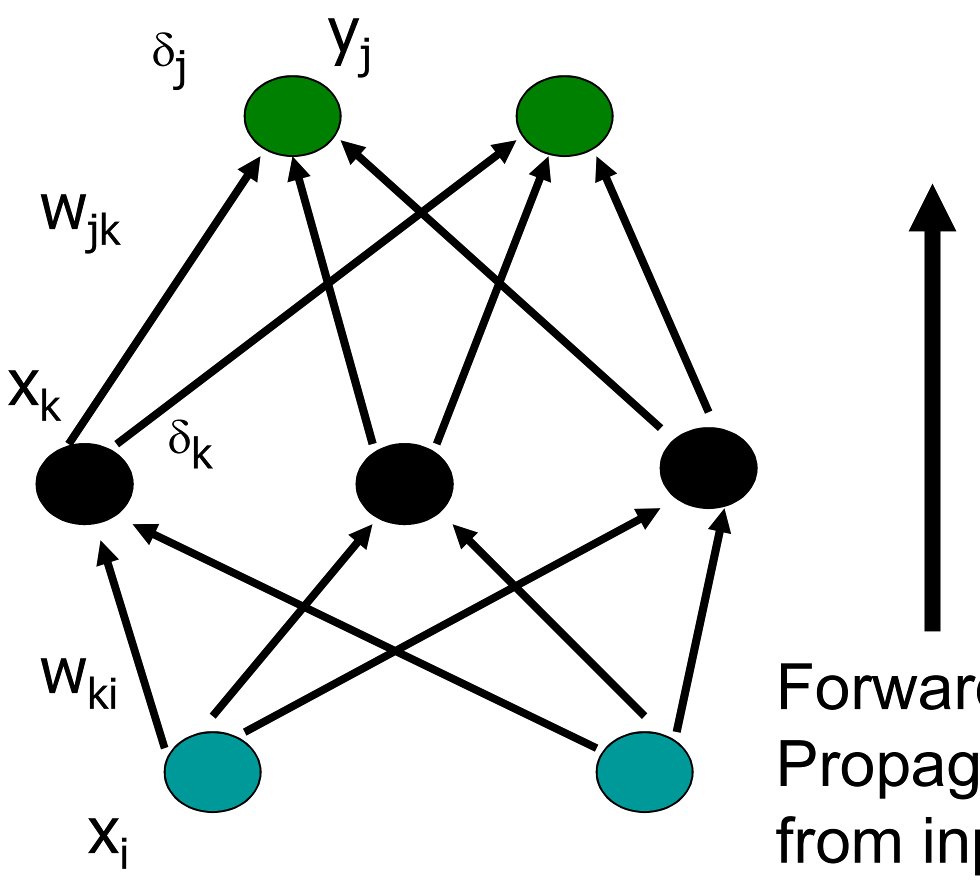
Then…

The problem is: ***How to train Multi Layer Perceptrons??***

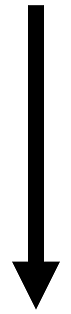
Solution: Backpropagation Algorithm (Rumelhart and colleagues,1986)

Backpropagation

Rumelhart, Hinton and Williams (1986) (**though actually invented earlier in a PhD thesis relating to economics)**



Backward step:
propagate errors from output to hidden layer

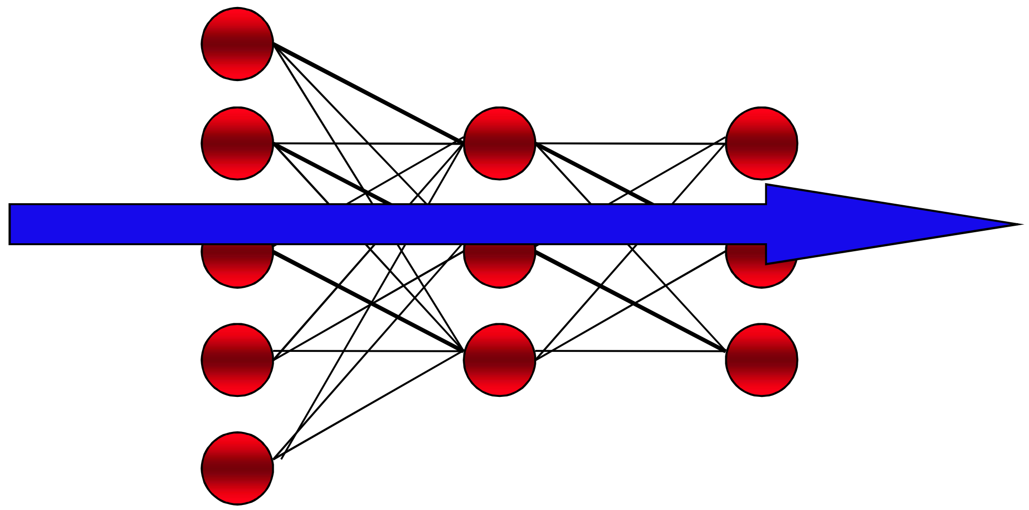


Forward step:
Propagate activation from input to output layer

Training MLPs

<u>Forward Pass</u>

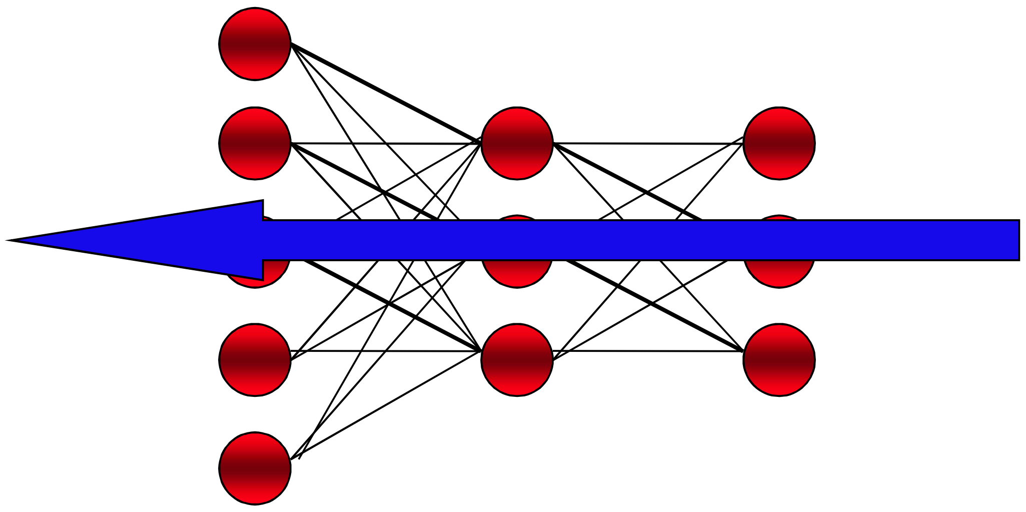
1. Put the input values in the input layer.
2. Calculate the activations of the hidden nodes.
3. Calculate the activations of the output nodes.



Training MLPs

Backward Pass

1. Calculate the output errors
2. Update last layer of weights.
3. Propagate error backward, update hidden weights.
4. Until first layer is reached.



Error Function

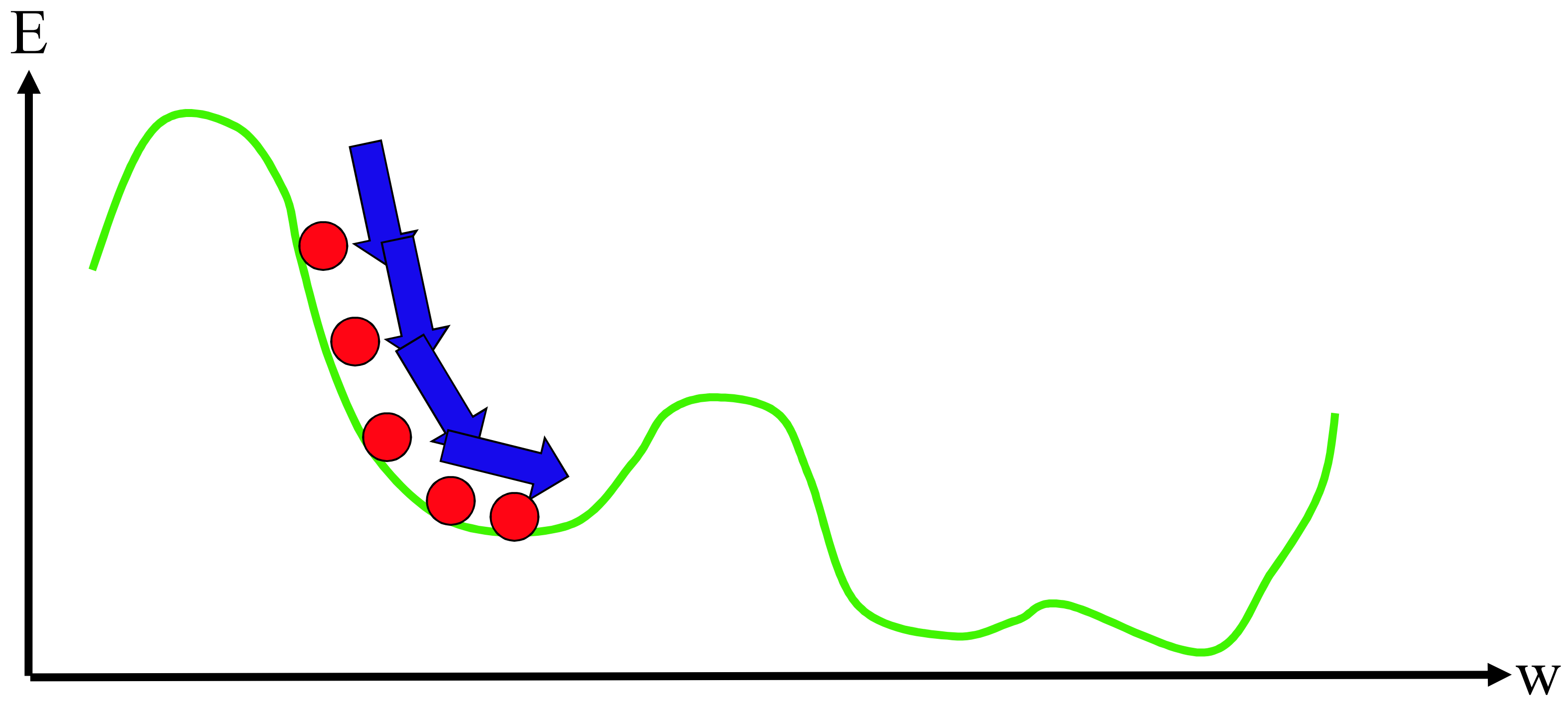
- Single scalar function for entire network.
- Parameterized by weights (objects of interest).
- Multiple errors of different signs should not cancel out.
- ***Sum-of-squares error:***

$$E(\mathbf{w}) = \frac{1}{2}\sum_k (t_k - y_k)^2 = \frac{1}{2}\sum_k \left(t_k - \sum_i w_{ik}x_i\right)^2$$

Back Propagation Algorithm

- The backpropagation training algorithm uses the ***gradient descent*** technique to **minimize the *mean square difference*** between the desired and actual outputs.
- The network is trained initially selecting ***small random weights*** and then presenting all training data incrementally.
- ***Weights*** are ***adjusted*** after every trial until they ***converge*** and the error is reduced to an acceptable value.

Gradient Descent



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

Error Terms

- Need to differentiate the **error function**
- The full calculation is presented in the book.
- Gives us the following *error terms* (deltas)
  - For the outputs

$$\delta_k = (y_k - t_k) g'(a_k)$$

  - For the hidden nodes

$$\delta_i = g'(u_i) \sum_k \delta_k w_{ik}$$

Update Rules

- This gives us the necessary update rules
  - For the weights connected to the outputs:

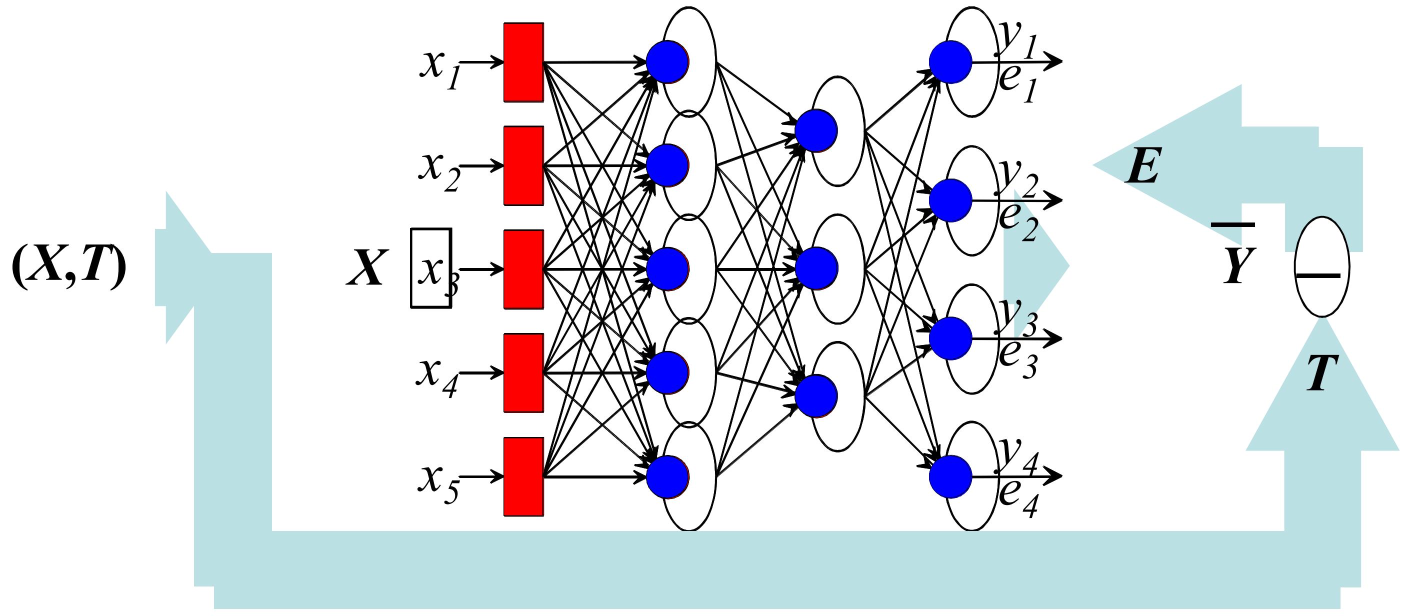
$$w_{jk} \leftarrow w_{jk} - \eta \delta_k z_j$$

  - For the weights on the hidden nodes:

$$v_{ij} \leftarrow v_{ij} - \eta \delta_j x_i$$

  - The learning rate η depends on the application. Values between 0.1 and 0.9 have been used in many applications.

BackPropagation Algorithm



Algorithm (sequential)

1. Apply an input vector and calculate all activations, *a* and *u*
2. Evaluate deltas for all output units:

$$\delta_k = (y_k - t_k)g'(a_k)$$

3. Propagate deltas backwards to hidden layer deltas:

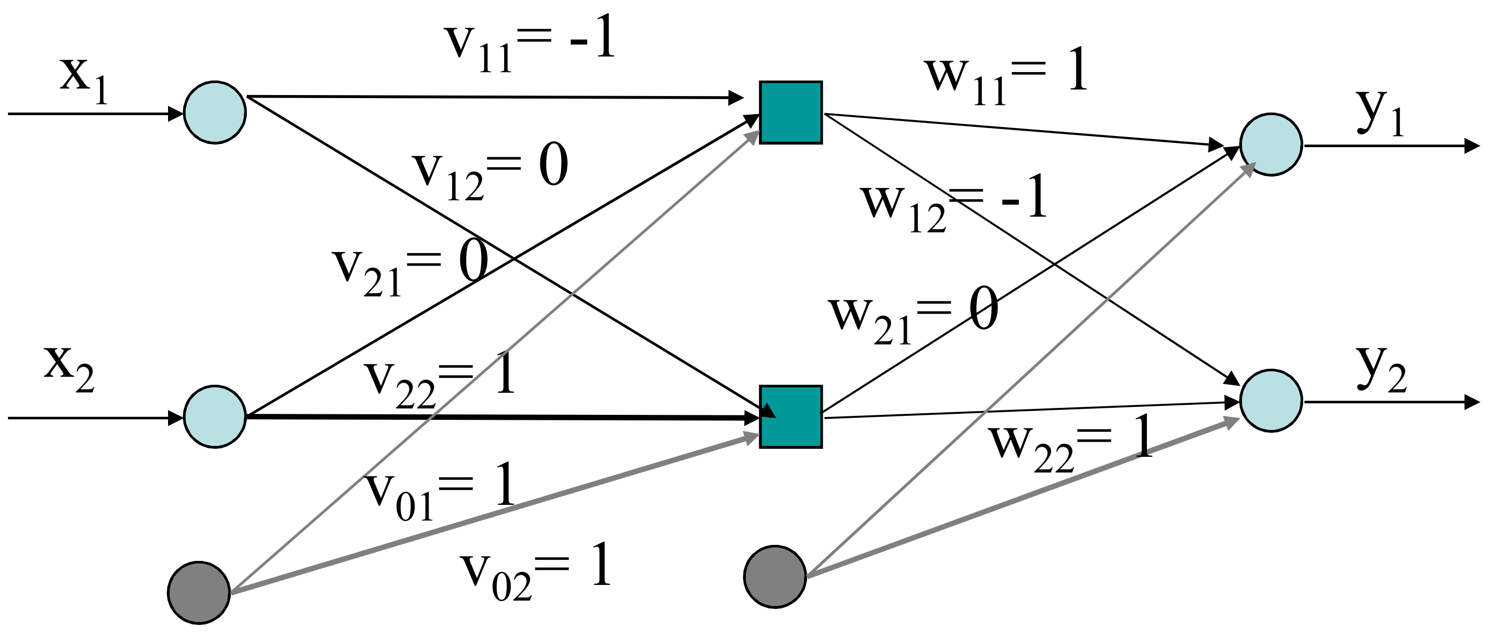
$$\delta_i = g'(u_i)\sum_k \delta_k w_{ik}$$

4. Update weights:

$$w_{jk} \leftarrow w_{jk} - \eta\delta_k z_j$$

$$v_{ij} \leftarrow v_{ij} - \eta\delta_j x_i$$

Example: Backpropagation

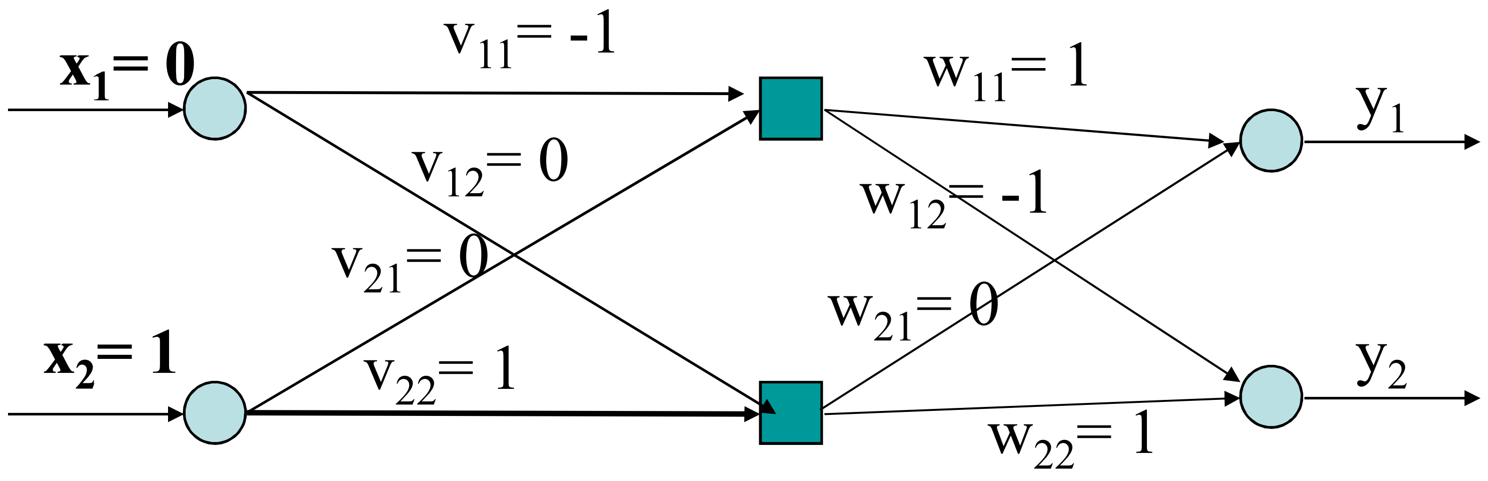


Use identity activation function (ie g(a) = a) for simplicity of example

Example: Backpropagation

All biases set to 1. Will not draw them for clarity.

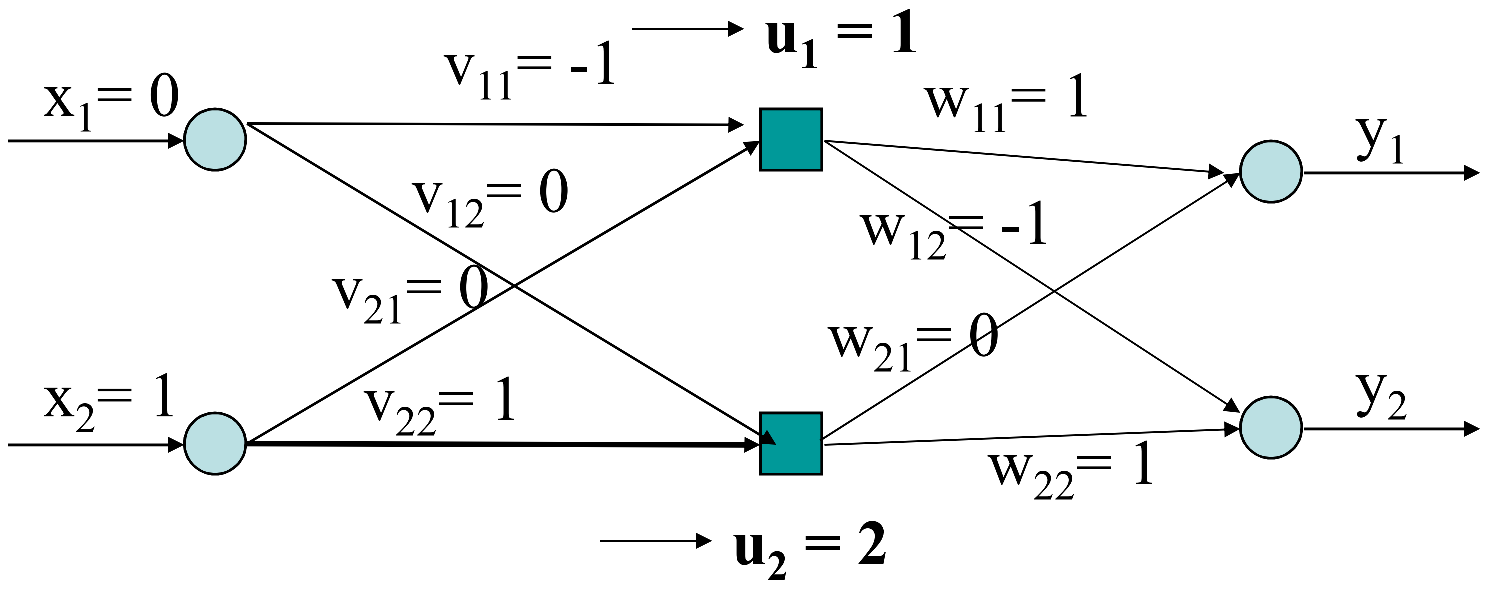
Learning rate h = 0.1



Have input [0 1] with target [1 0].

Example: Backpropagation

Forward pass. Calculate 1st layer activations:

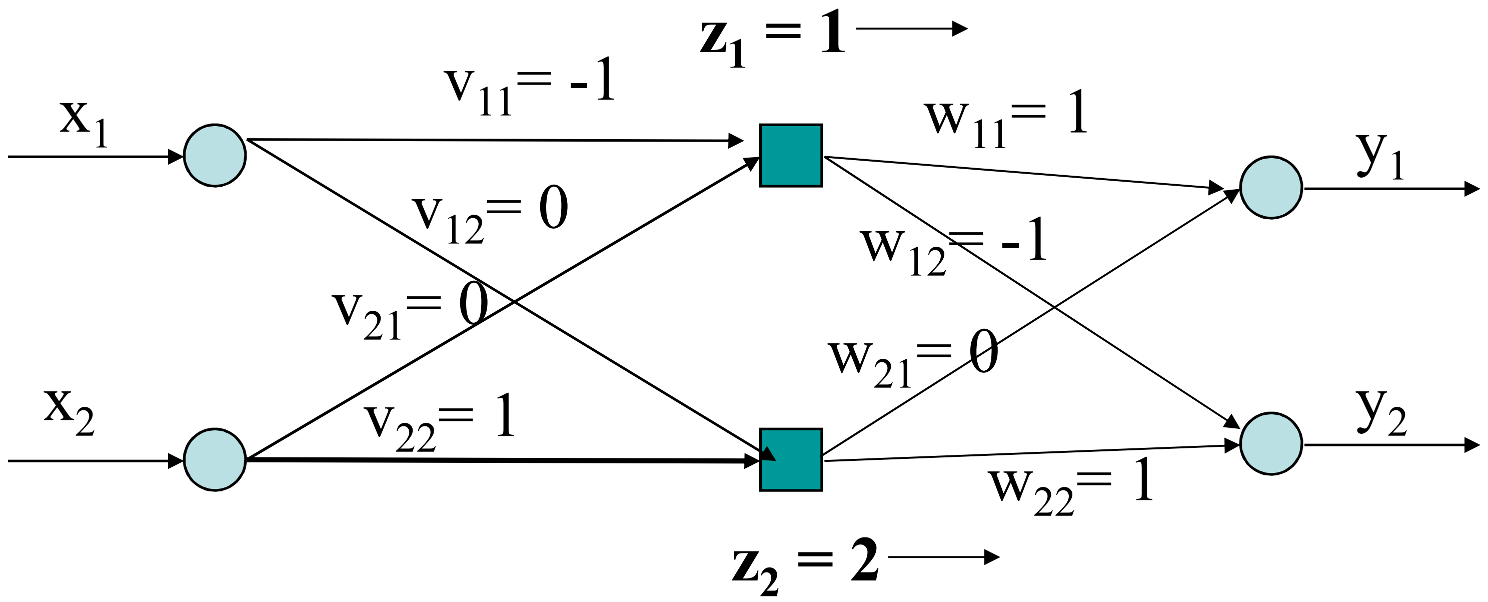


$u_1 = -1x0 + 0x1 + 1 = 1$

$u_2 = 0x0 + 1x1 + 1 = 2$

Example: Backpropagation

Calculate first layer outputs by passing activations through activation functions

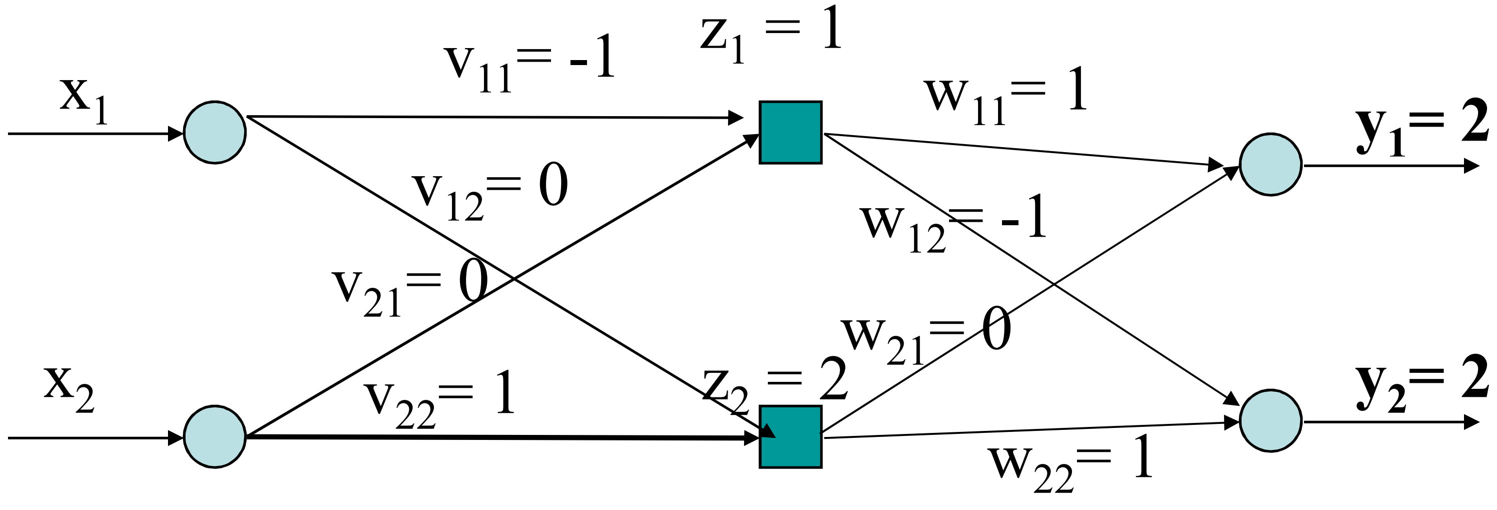


$$z_1 = g(u_1) = 1$$

$$z_2 = g(u_2) = 2$$

Example: Backpropagation

Calculate 2nd layer outputs (weighted sum through activation functions):



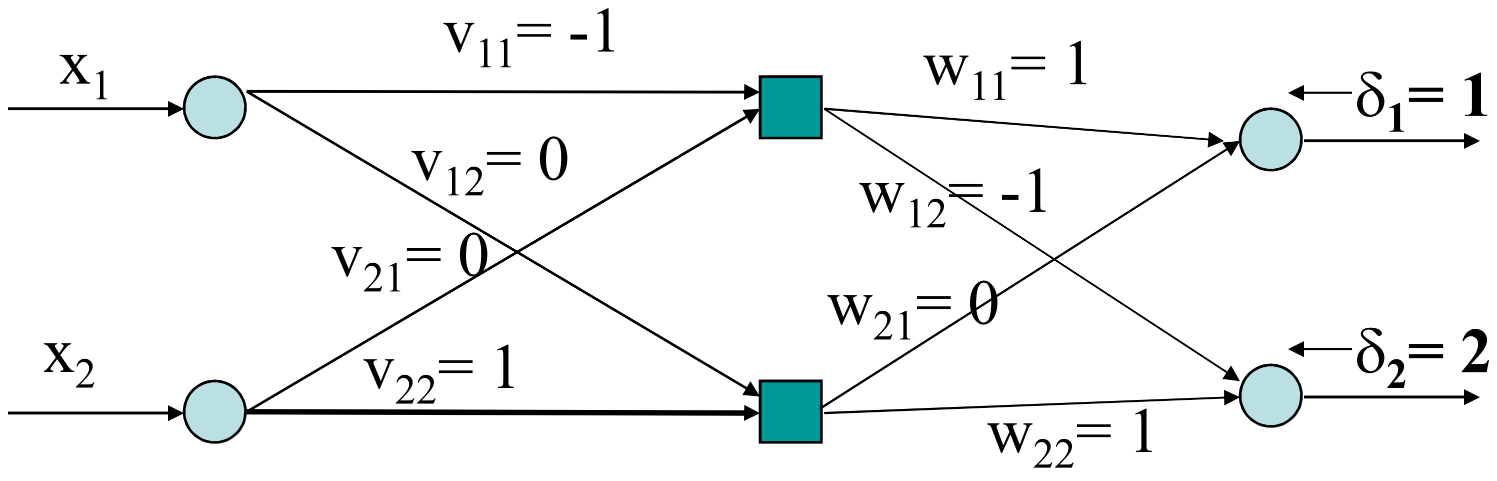
$$y_1 = a_1 = 1x1 + 0x2 +1 = 2$$

$$y_2 = a_2 = -1x1 + 1x2 +1 = 2$$

Example: Backpropagation

Backward pass:

$$\delta_k = (y_k - t_k) g'(a_k)$$



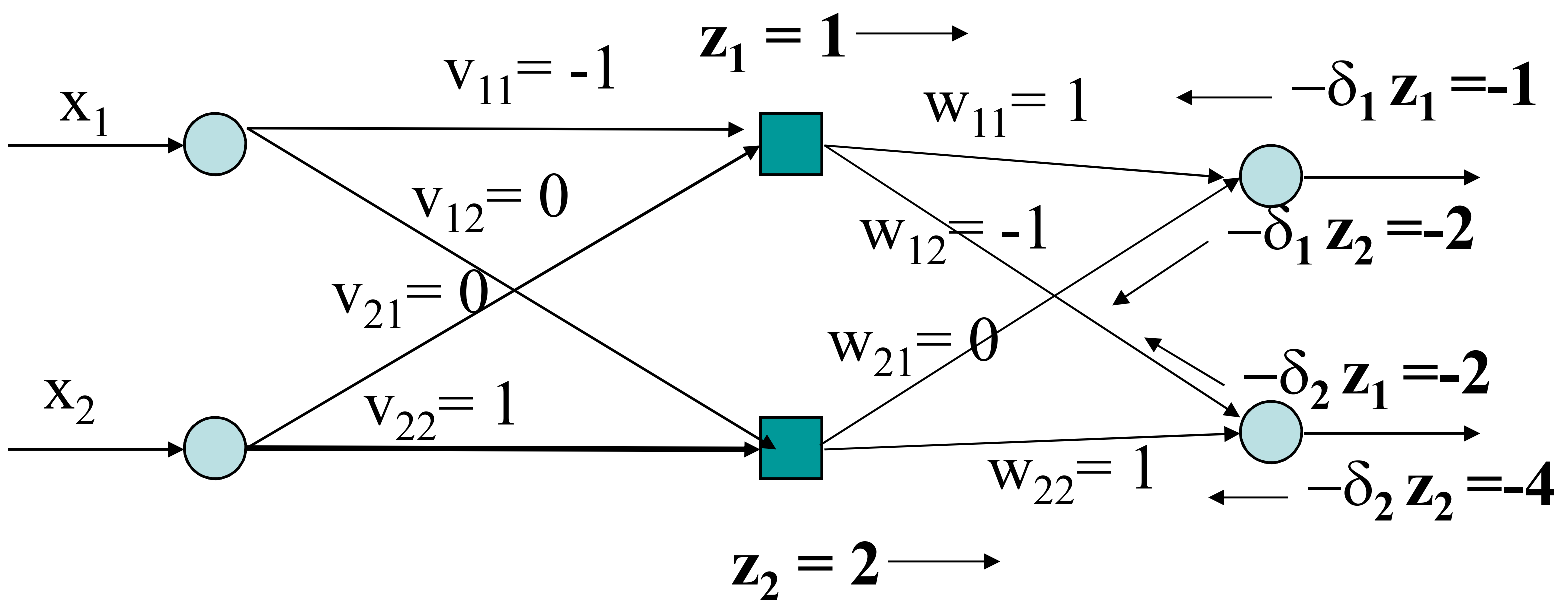
Target =[1, 0] so $t_1 = 1$ and $t_2 = 0$. So:

$\delta_1 = (y_1 - t_1) = 2 - 1 = 1$

$\delta_2 = (y_2 - t_2) = 2 - 0 = 2$

Example: Backpropagation

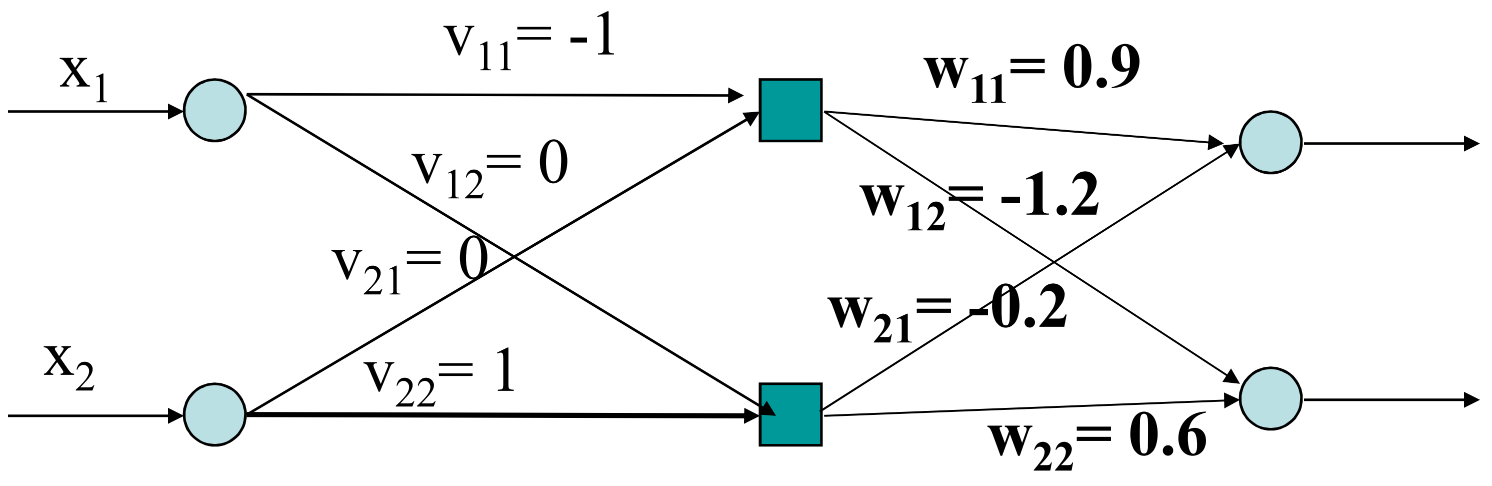
Calculate weight changes for 1st layer:



$$w_{jk} \leftarrow w_{jk} - \eta \delta_k z_j$$

Example: Backpropagation

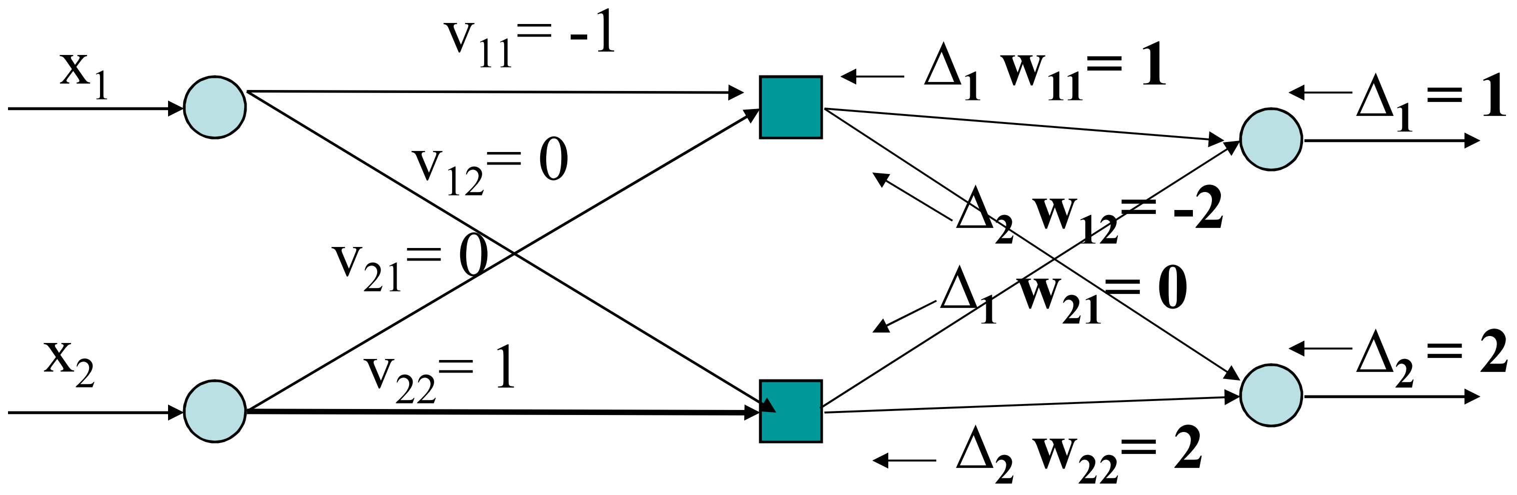
Weight changes will be:



$$w_{jk} \leftarrow w_{jk} - \eta \delta_k z_j$$

Example: Backpropagation

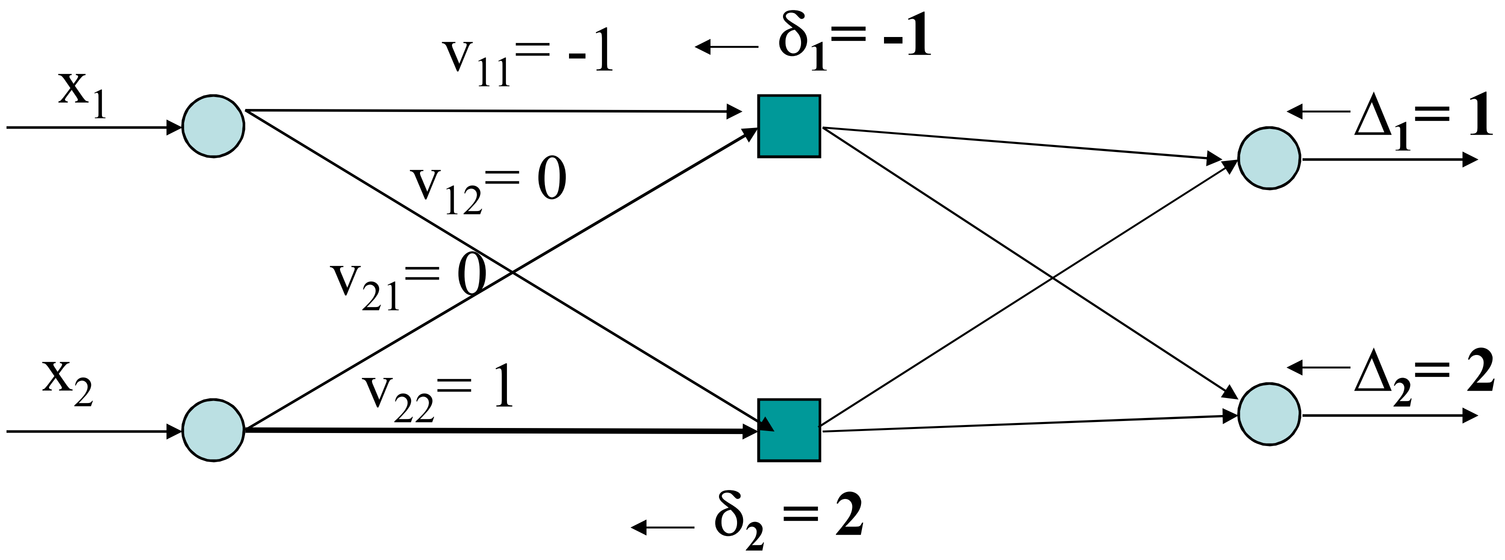
Calculate hidden layer deltas:



$$\delta_i = g'(u_i) \sum_k \Delta_k w_{ik}$$

Example: Backpropagation

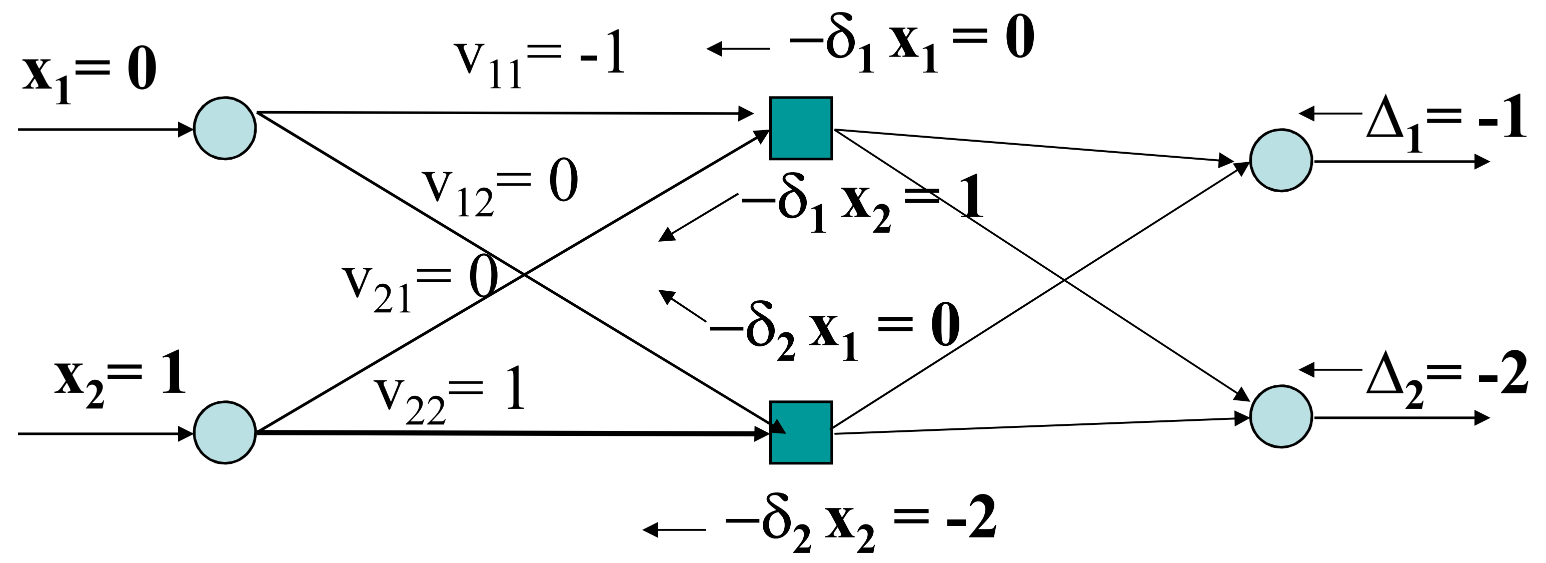
Deltas propagate back: $\delta_i = g'(u_i) \sum_k \Delta_k w_{ik}$



$$\delta_1 = 1 - 2 = -1$$
$$\delta_2 = 0 + 2 = 2$$

Example: Backpropagation

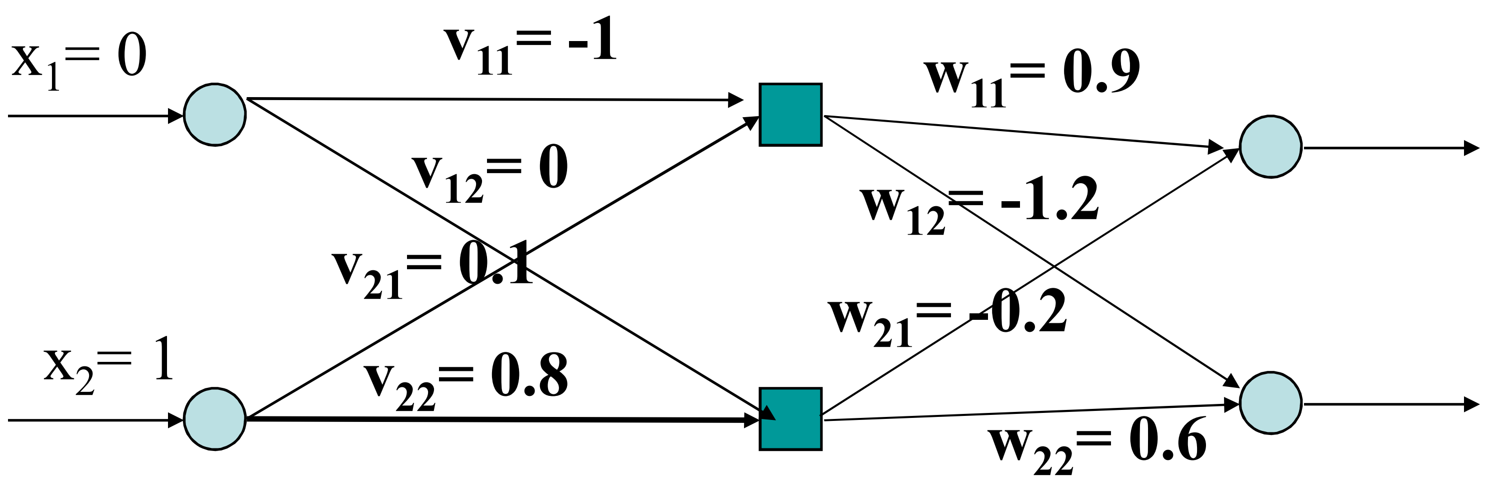
And are multiplied by inputs:



$$v_{ij} \leftarrow v_{ij} - \eta \delta_j x_i$$

Example: Backpropagation

Finally change weights: $$v_{ij} \leftarrow v_{ij} - \eta \delta_j x_i$$

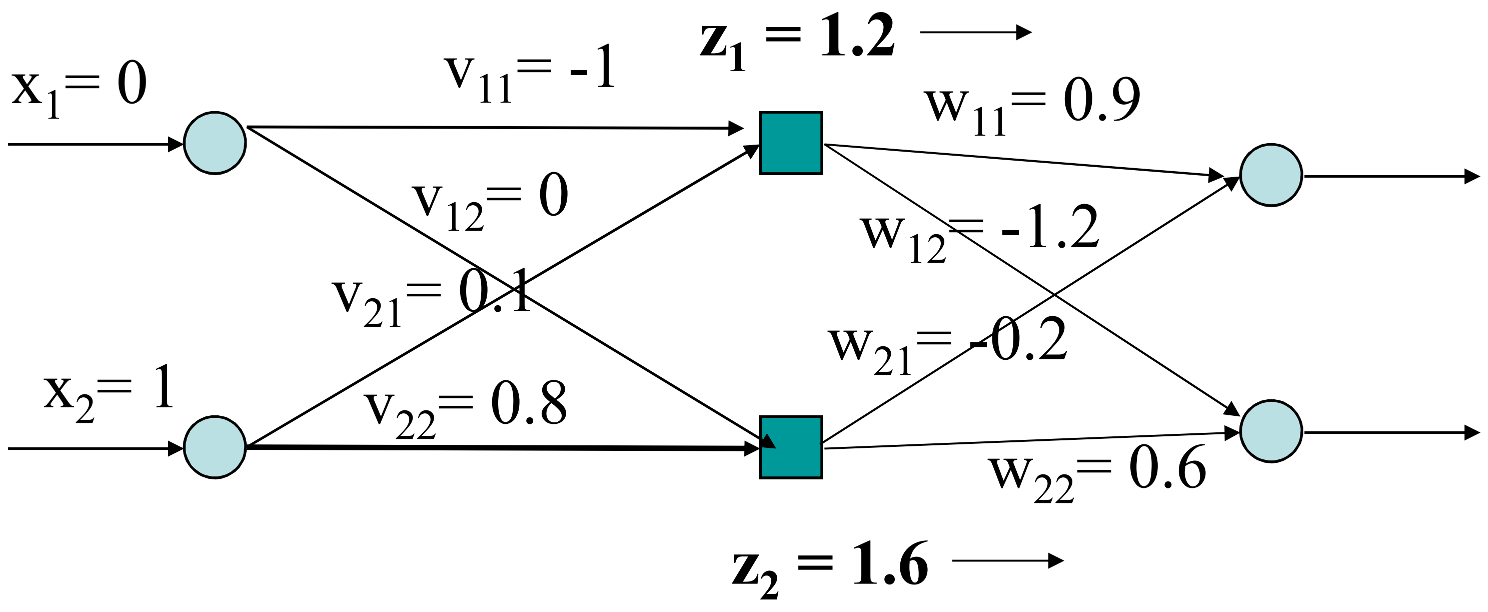


Note that the weights multiplied by the zero input are unchanged as they do not contribute to the error

We have also changed biases (not shown)

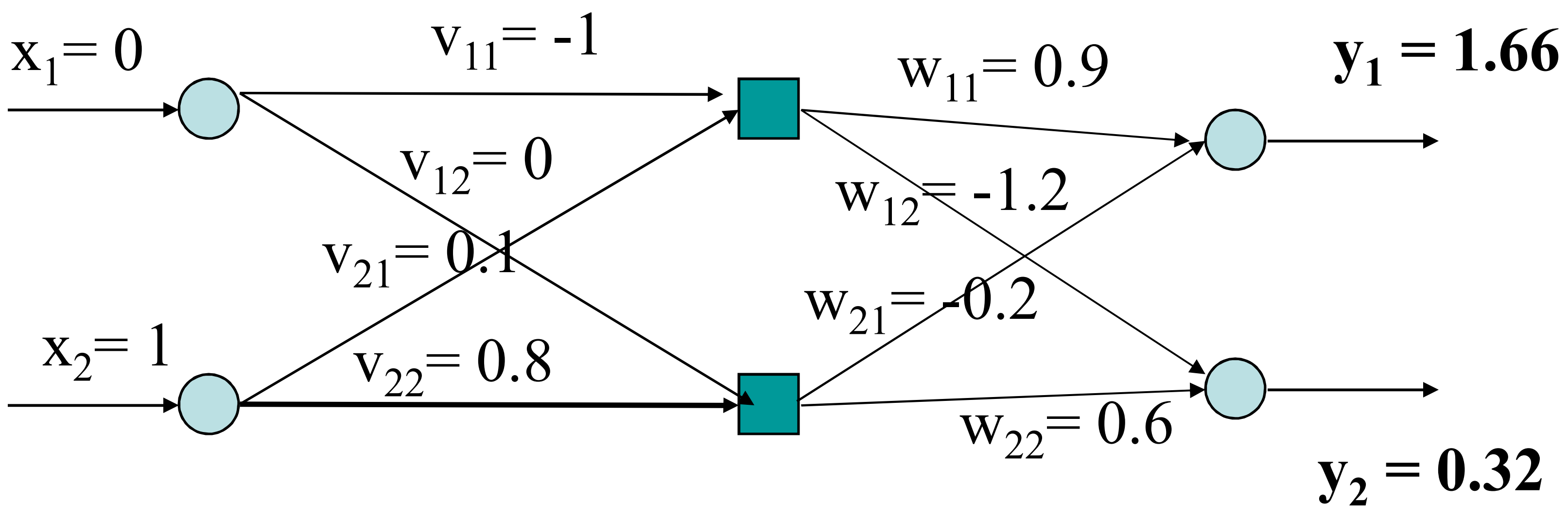
Example: Backpropagation

Now go forward again (would normally use a new input vector):



Example: Backpropagation

Now go forward again (would normally use a new input vector):

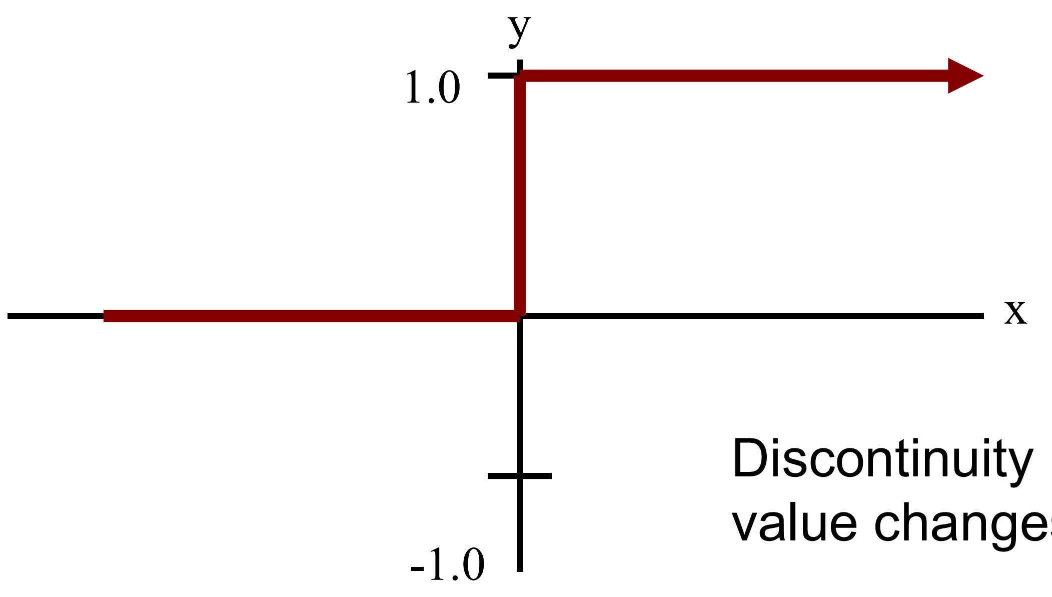


Outputs now closer to target value [1, 0]

Activation Function

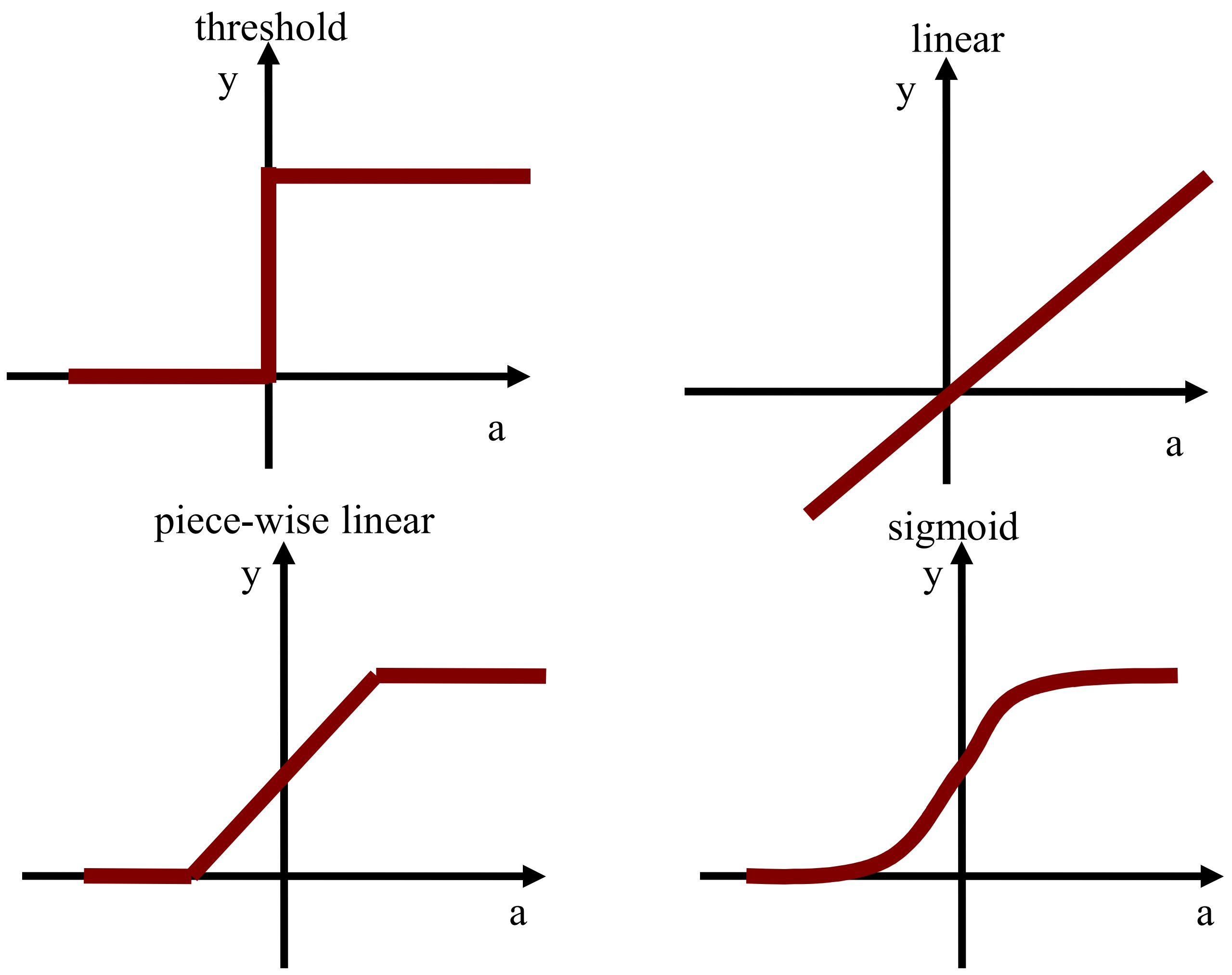
- We need to compute the derivative of activation function *g*
- What do we want in an activation function?
    - Differentiable
    - Nonlinear (more powerful)
    - Bounded range (for numerical stability)

Hard Limit Function



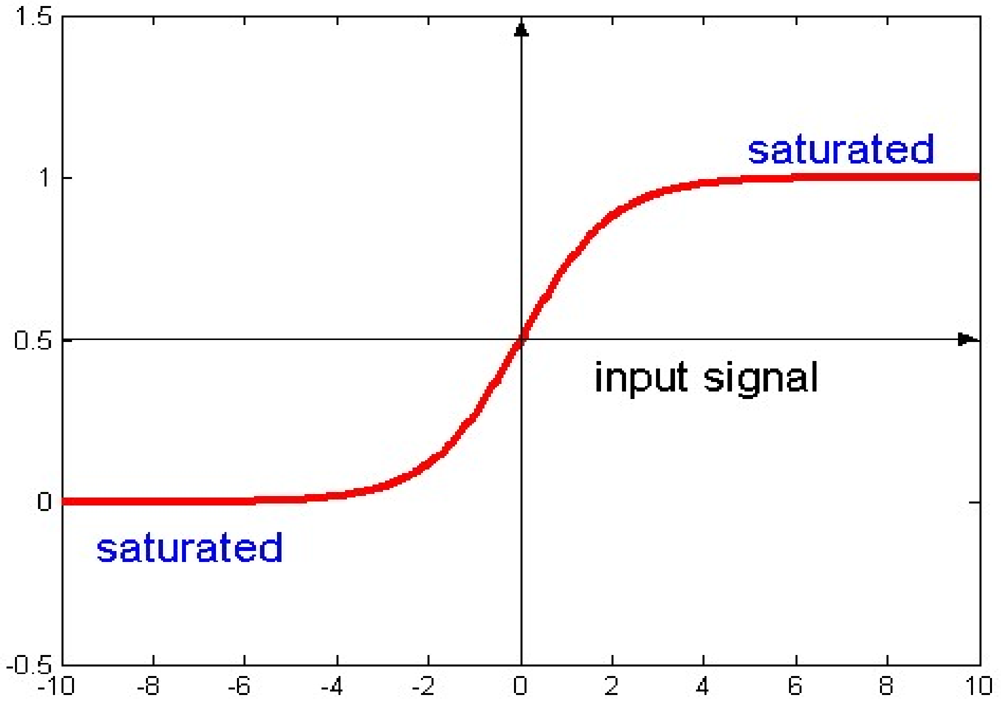
Discontinuity where the value changes from *0* to 1.

A Quick Overview (Activation Functions)



Sigmoidal Function - Common in MLP

$$g(a_i) = \frac{1}{1 + \exp(-k a_i)} = \frac{1}{1 + e^{-k a_i}}$$

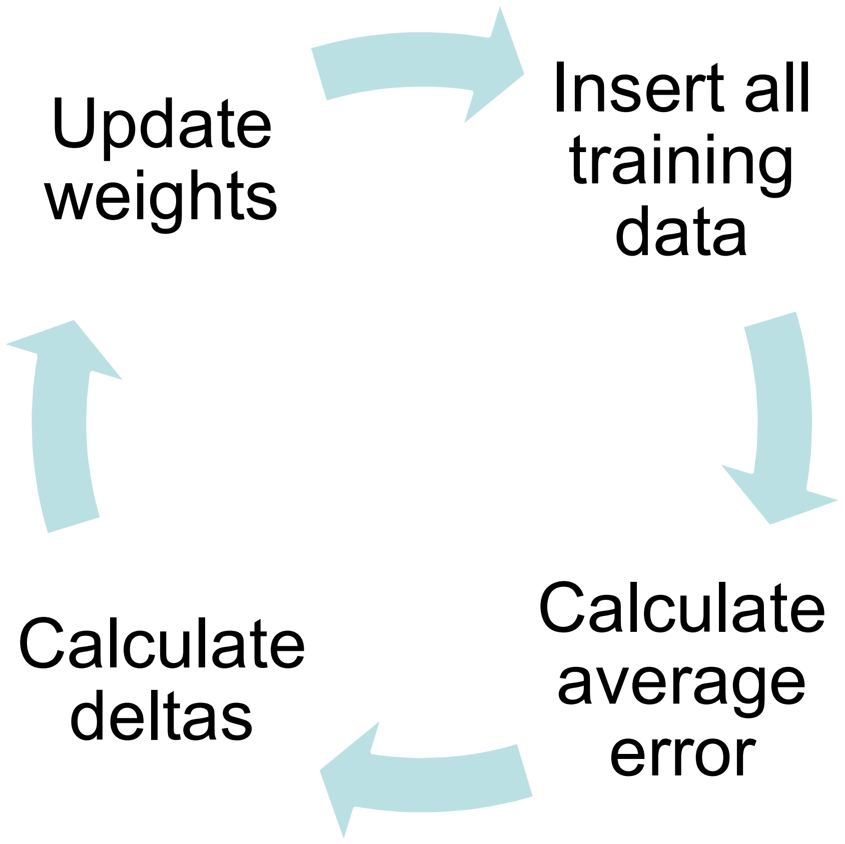


- Where *k* is a positive constant.
- The sigmoidal function gives a value in range of 0 to 1.
- Alternatively can use *tanh(ka)* which is same shape but in range –1 to 1.

Network Training

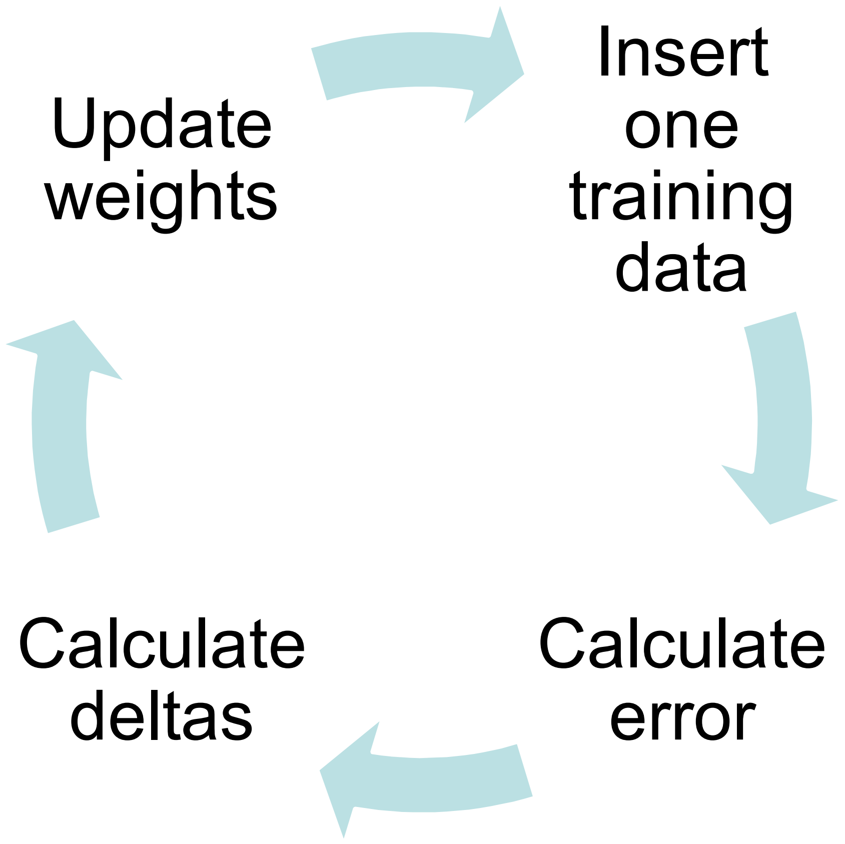
- Training set shown repeatedly until stopping criteria are met.
- ***When should the weights be updated?***
  - After all inputs seen (***batch***)
  - After each input is seen (***sequential***)
  - Both ways, **need many epochs** - passes through the whole dataset

Batch Training



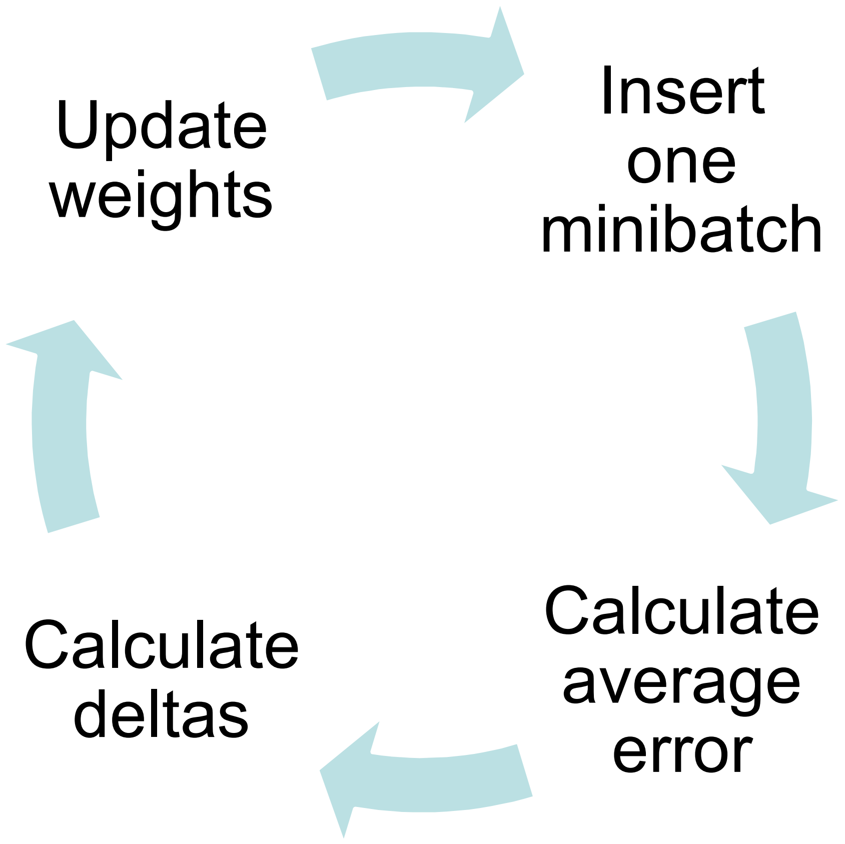
- One loop is called an **epoch**
- More accurate estimate of gradient
- Faster convergence to local optimum

Sequential Training



- Simpler to program
- Can avoid local optima

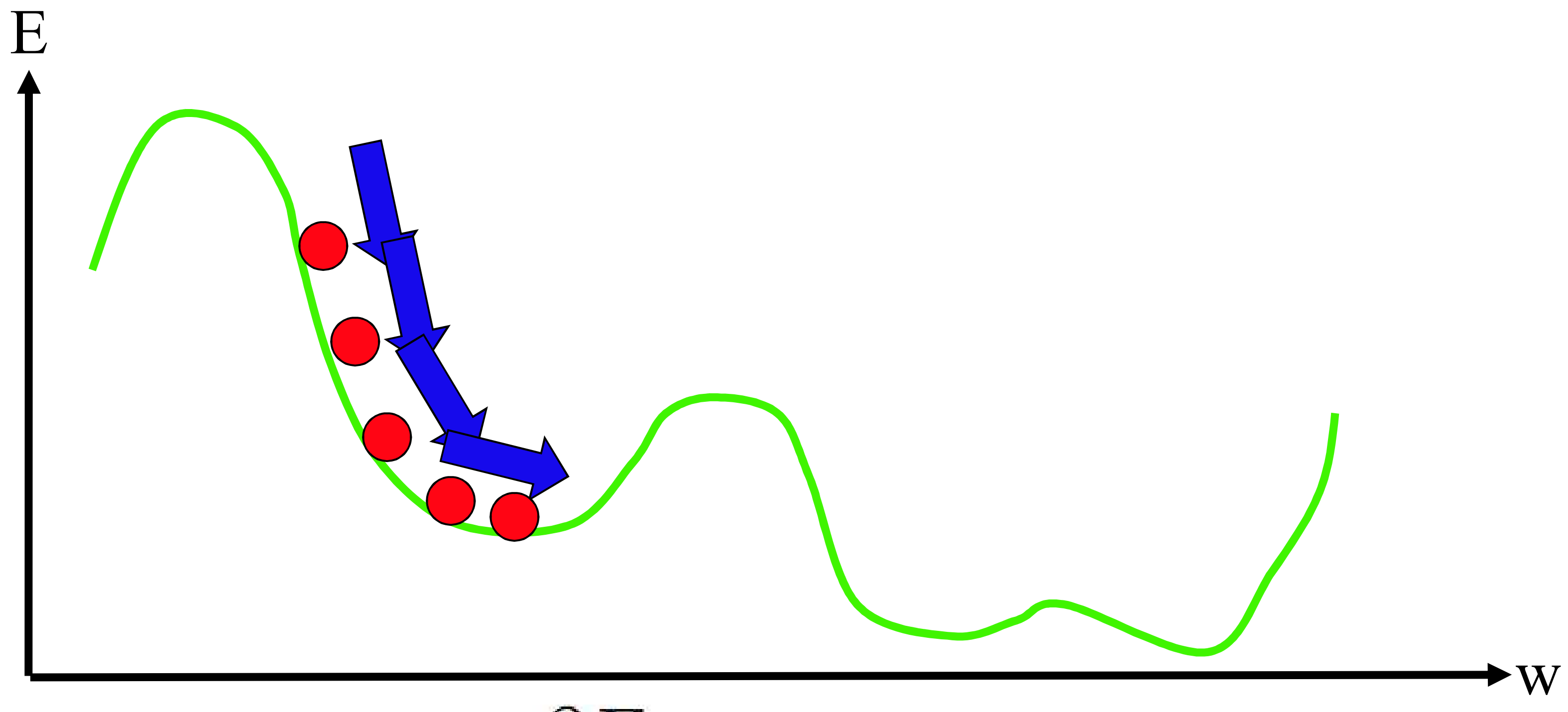
Compromise: Minibatch



- Split all data into N minibatches
- Feed each minibatch to the network
- Shuffle data and repat

- May lead to the benefits of both batch-learning and sequential learning:
  - Rapid convergence
  - Avoiding local minima

Risk: Gradient descent takes us to local minimum

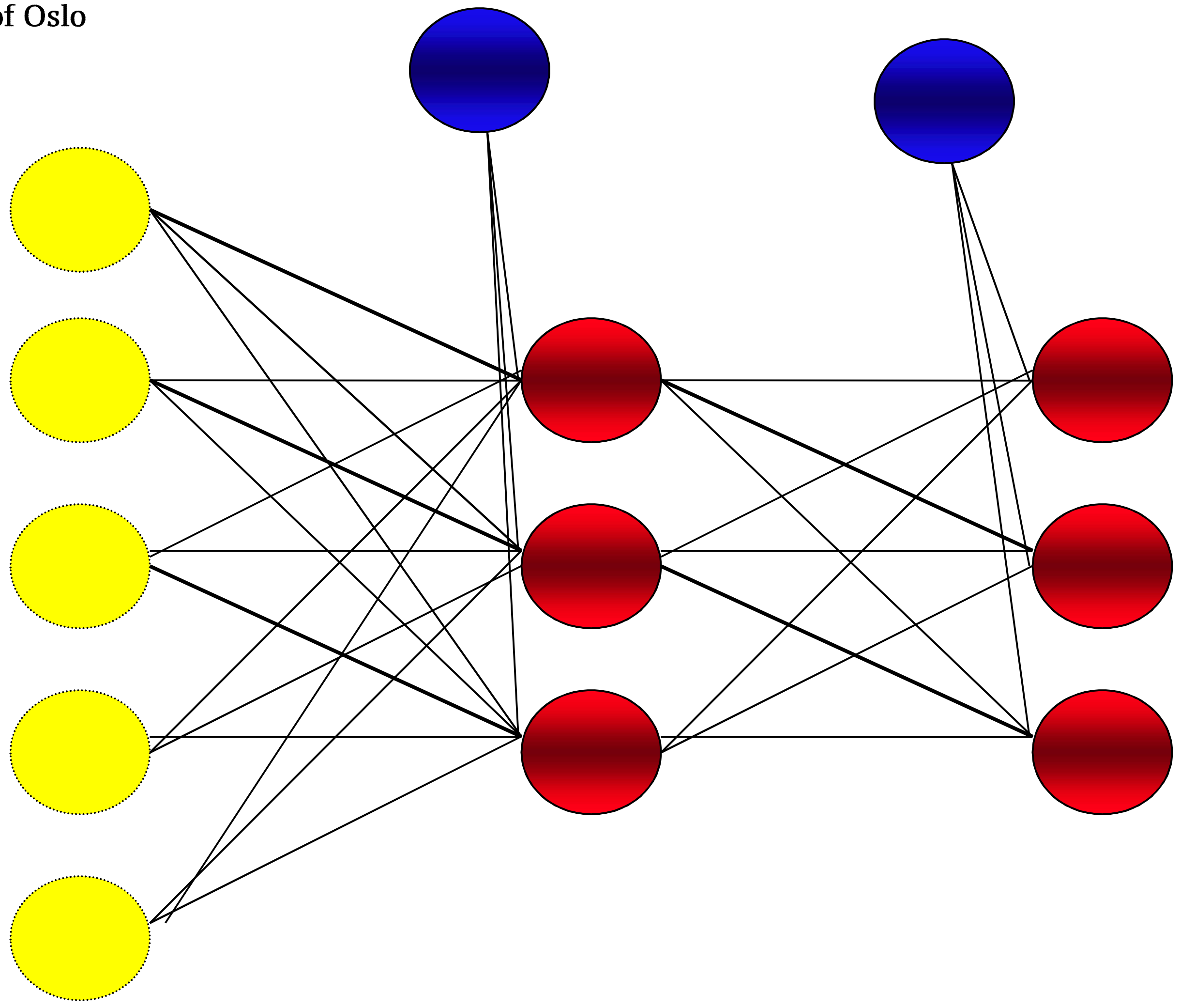


$$\Delta w_{ik} = -\eta \frac{\partial E}{\partial w_{ik}}$$

How can we avoid the local minimum?

- Initialize training many times with random weights
- Use **momentum:**

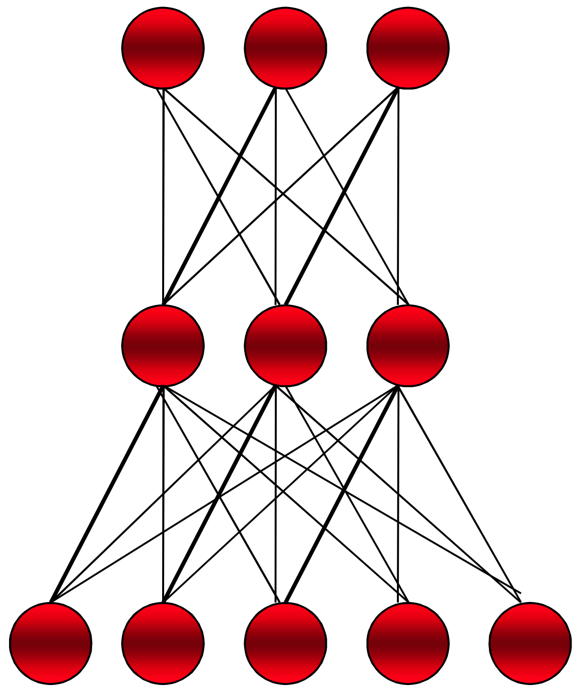
$$w_{ij} \leftarrow w_{ij} - \eta \Delta_j z_i + \alpha \Delta w_{ij}^{t-1}$$



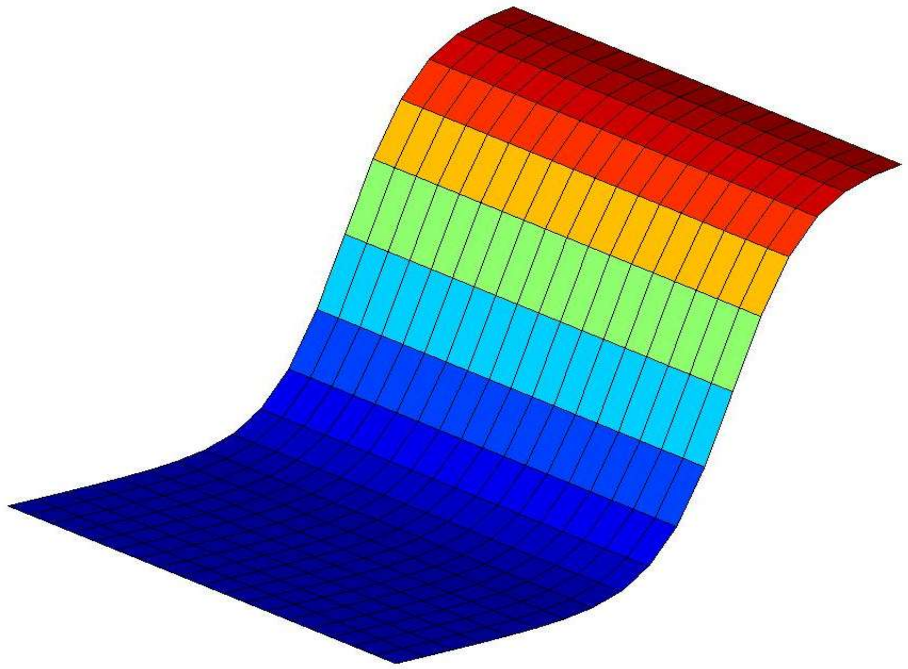
PRACTICAL ISSUES

Amount of Training

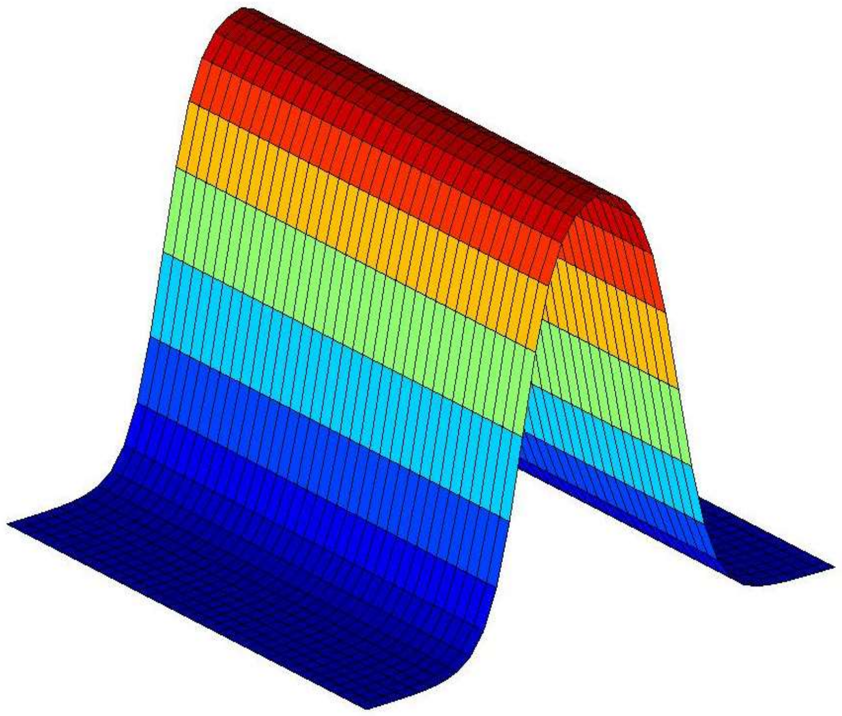
- How much training data is needed?
  - Count the weights
  - Rule of thumb: use 10 times more data than the number of weights



How many hidden layers do we need?



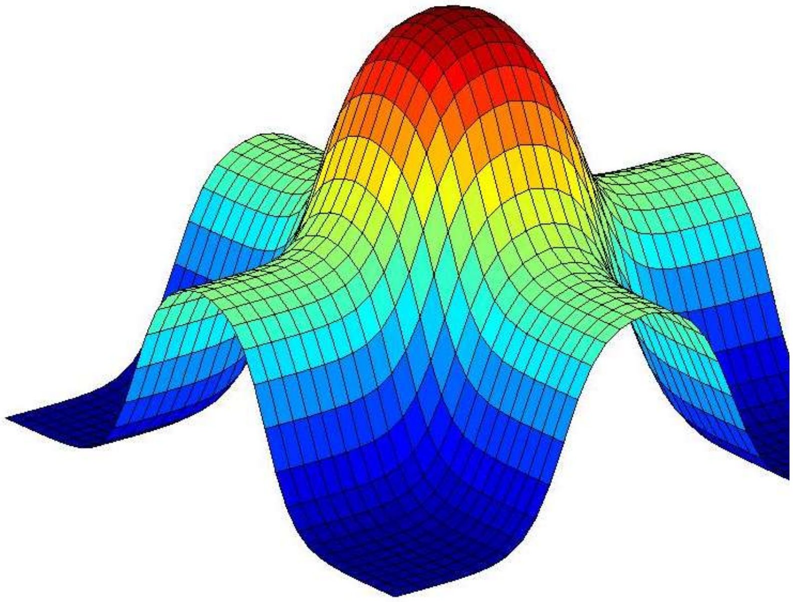
Output of one sigmoid

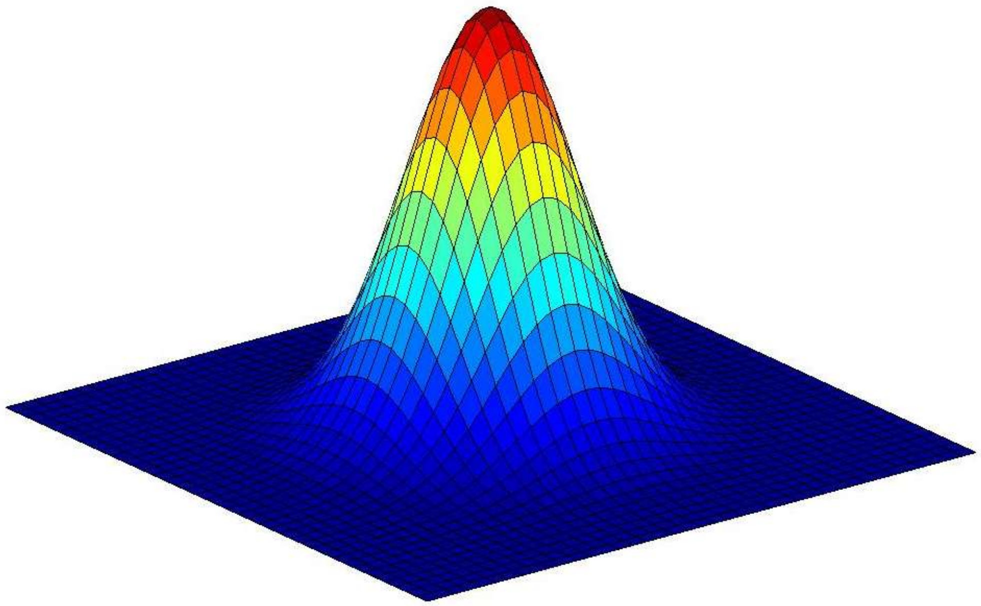


Addition of two sigmoids

How many hidden layers do we need?

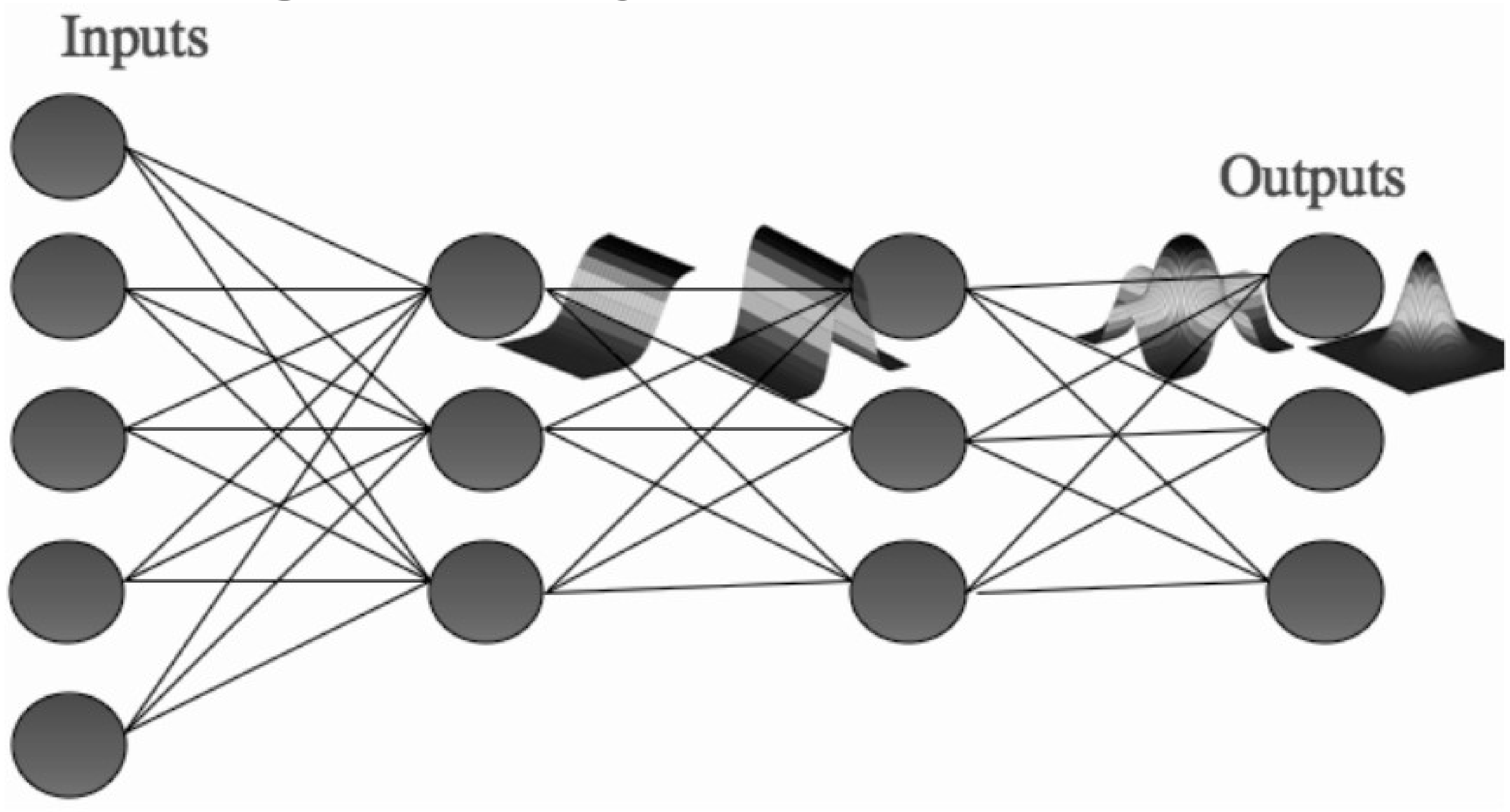
Addition of more ridges:
Localised response





Addition of two ridges
Unique maximum

Learning Capacity

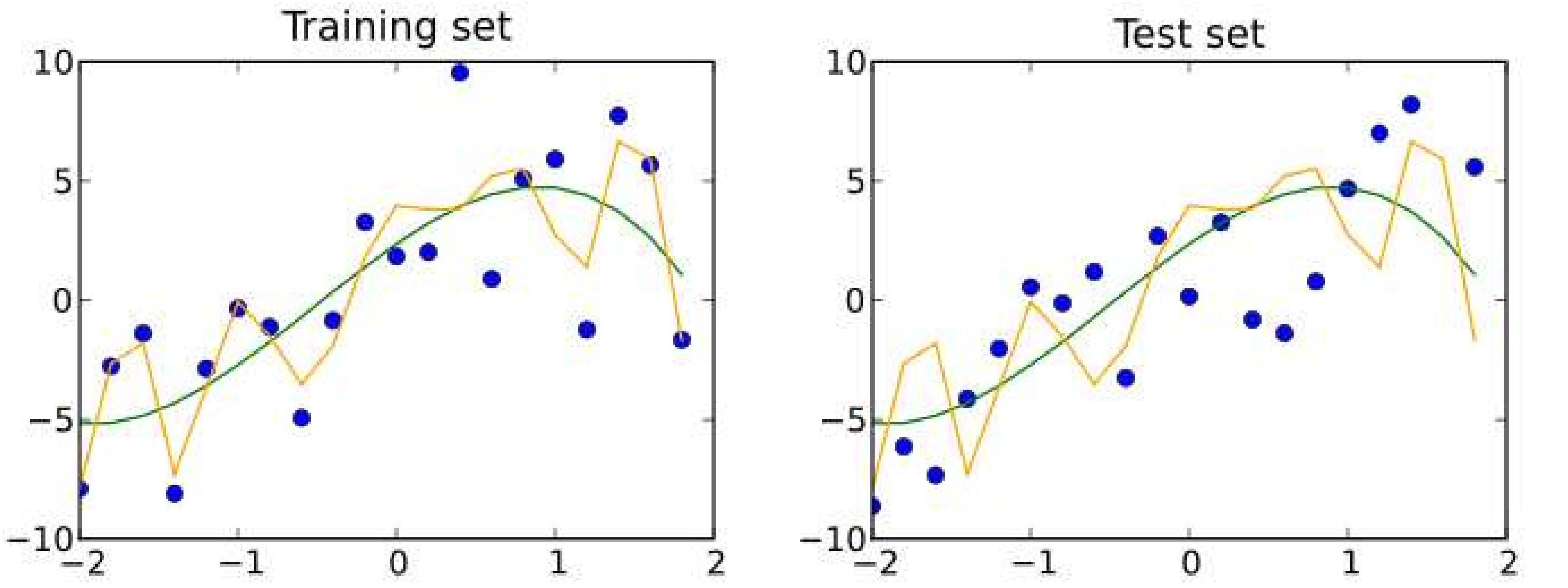


Universal approximation theorem: Any continuous function can be approximated by a neural network with a single hidden layer

Network Topology

- How many layers?
- How many neurons per layer?
- No good answers
  - At most 3 weight layers, usually 2
  - Test several different networks

- Possible types of adaptive algorithms (not default in MLP):
  - start from a large network and successively remove some neurons and links until network performance degrades.
  - begin with a small network and introduce new neurons until performance is satisfactory.



GENERALIZATION, TRAINING AND TESTING

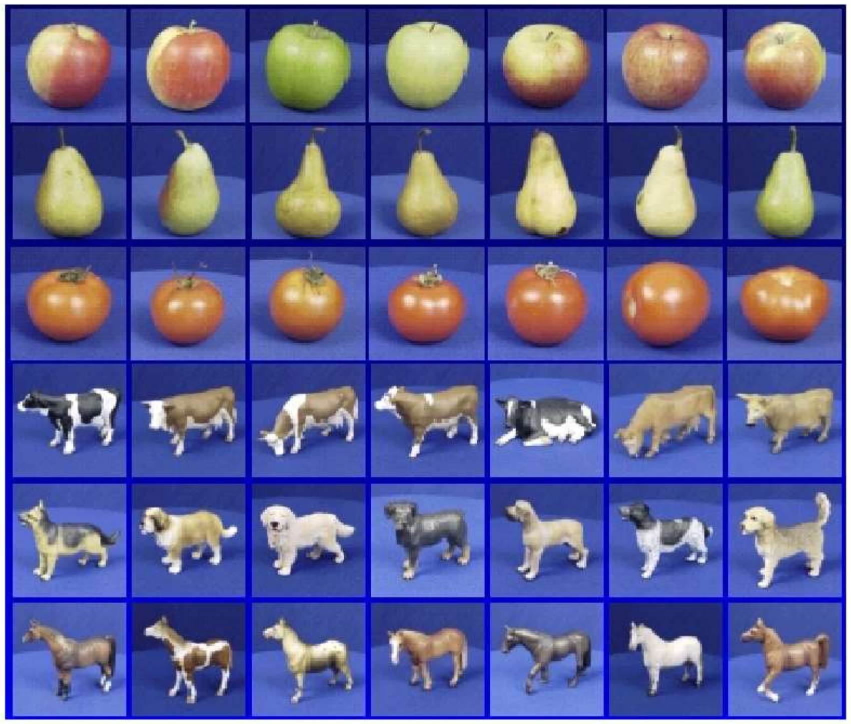
Generalisation

- Aim of neural network learning:
  - ***Generalise from training examples to all possible inputs.***

- The objective of learning is to achieve good ***generalization*** to new cases; we cannot train on all possible data.

- Under-training is ***bad***.
- Over-training is also ***bad.***

Generalisation - example



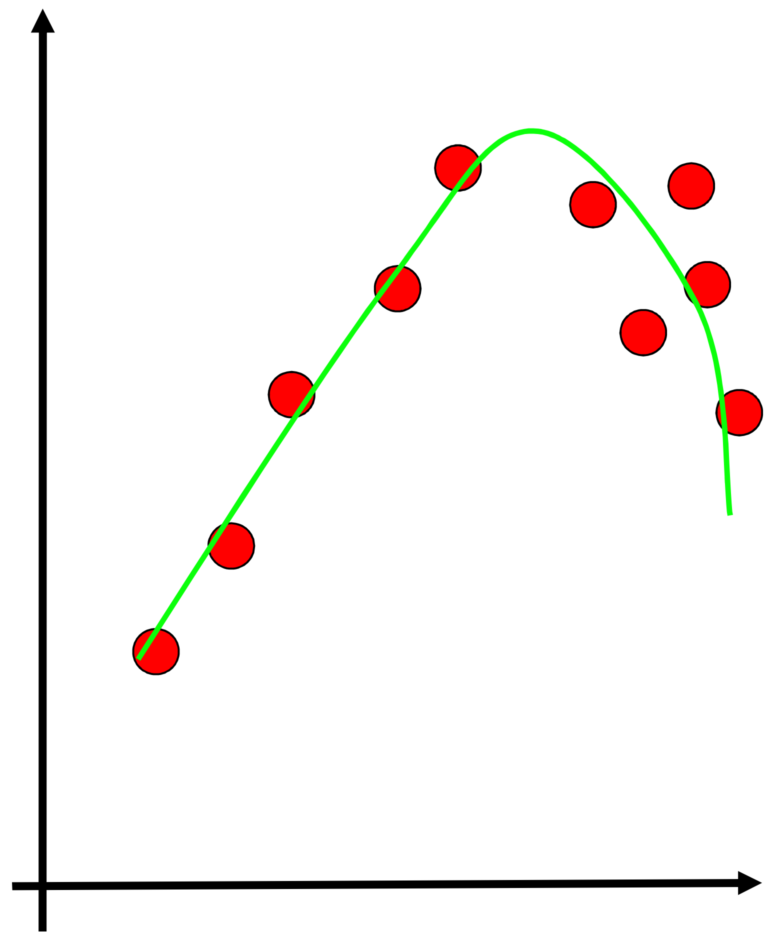
Given: training images and their categories

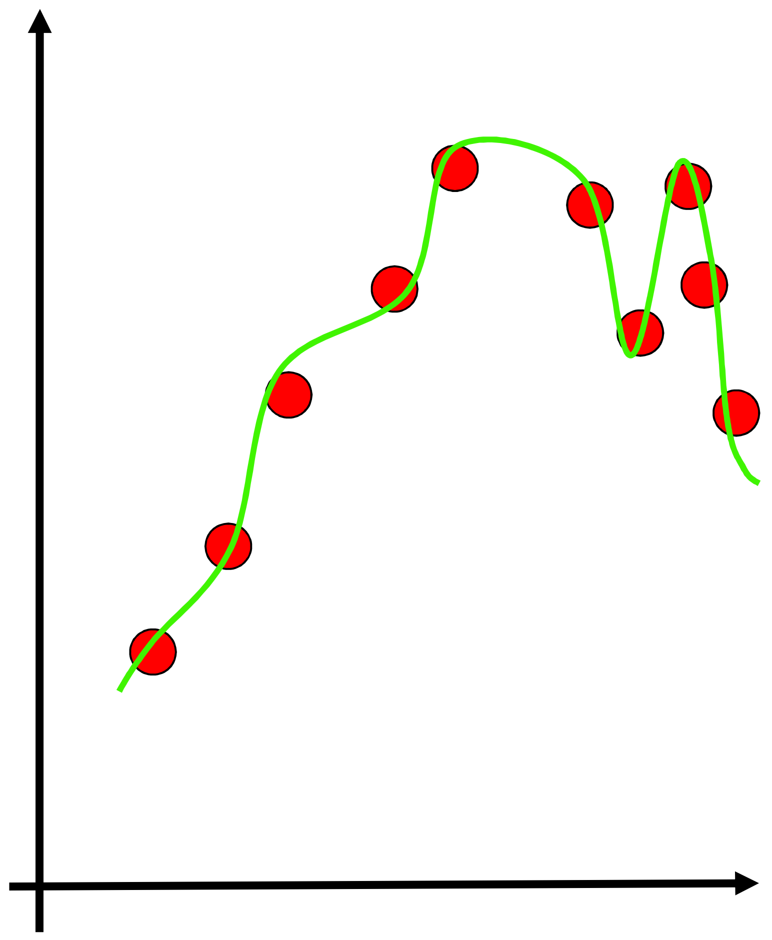
What are the categories of these test images?

Overfitting

- Overfitting occurs when a model begins to learn the **bias** of the training data rather than learning to generalize.
- Overfitting generally occurs when a model is excessively complex in relation to the amount of data available.
- A model which overfits the training data will generally have poor predictive performance, as it can exaggerate minor fluctuations in the data.

Overfitting



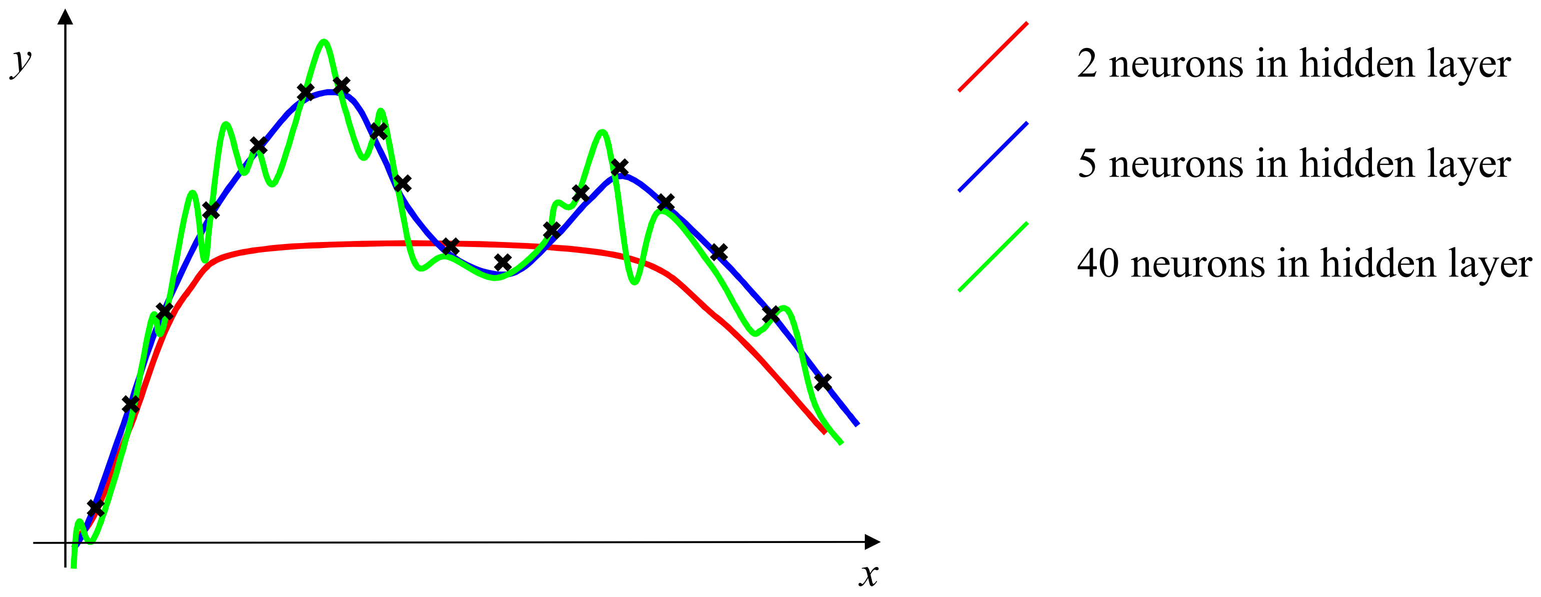


Overfitting

- The training data contains information about the regularities in the mapping from input to output.
- Training data also contains **bias**:
  - There is ***sampling bias***. There will be accidental regularities due to the finite size of the training set.
  - The target values may also be unreliable or noisy.
- When we fit the model, it cannot tell which regularities are relevant and which are caused by sampling error.
  - So it fits both kinds of regularity.
  - If the model is very flexible it can model the sampling error really well. ***This is not what we want***.

The Problem of Overfitting

- Approximation of the function $y = f(x)$:



The Solution: Cross-Validation

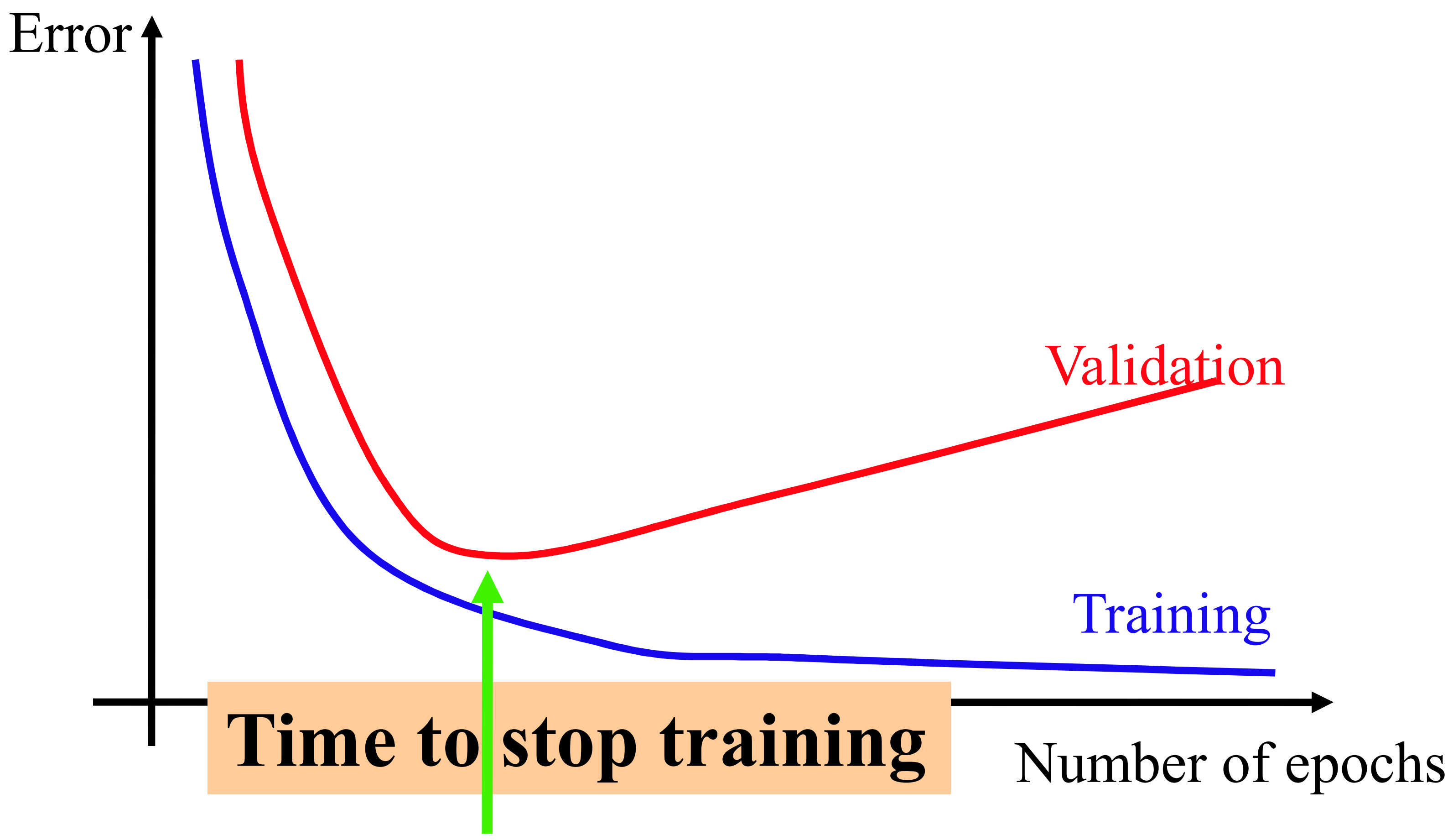
To maximize generalization and avoid overfitting, split data into three sets:

- **Training set**: Train the model.
- **Validation set**: Judge the model's generalization ability during training.
- **Test set**: Judge the model's generalization ability after training.

Validation set

- Data unseen by training algorithm – not used for backpropagation.
- Network is not trained on this data, so we can use it to measure generalization ability.
- Goal is to maximize generalization ability, so we should minimize the error on this data set.

Early Stopping

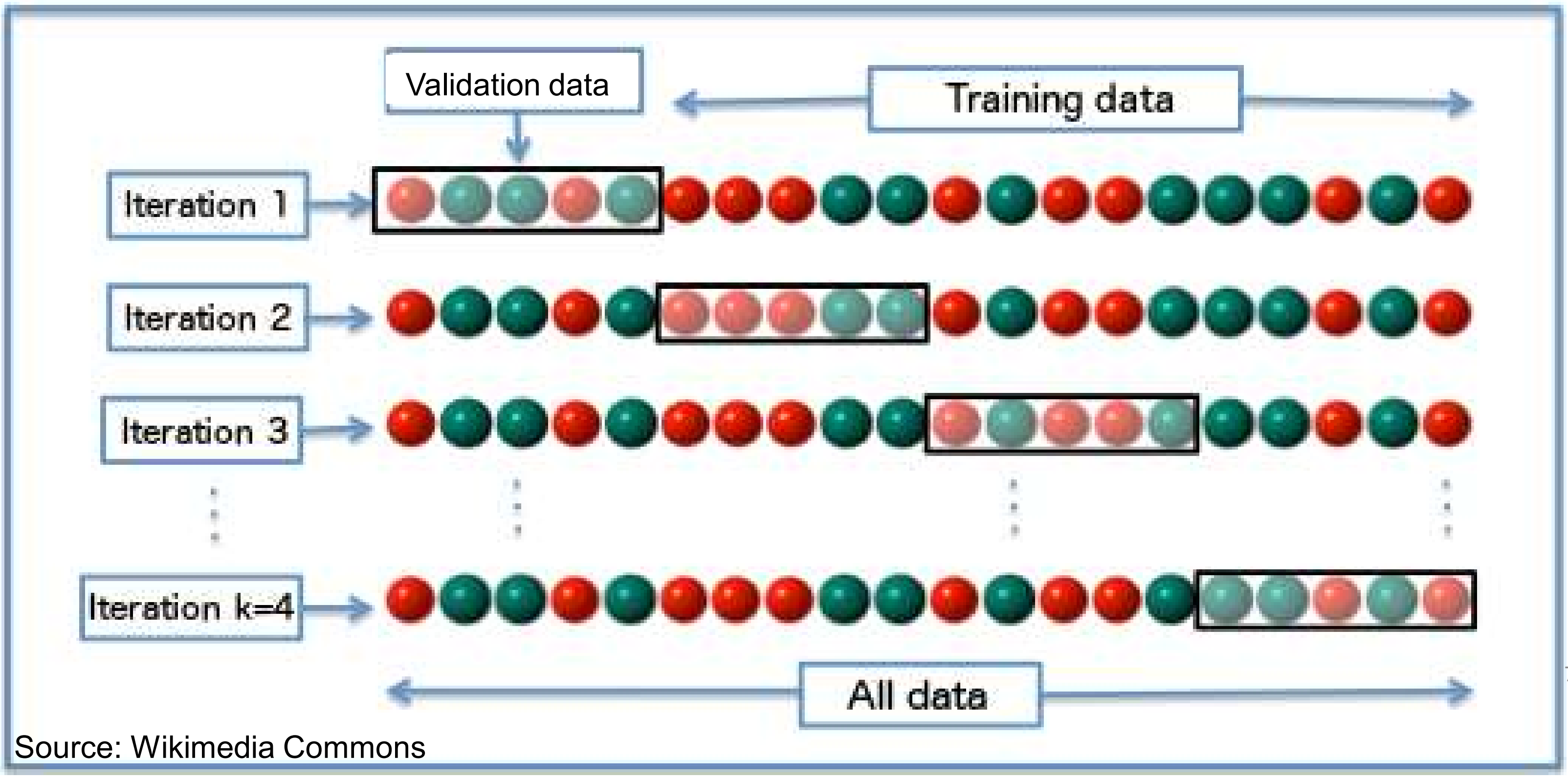


Testing set

- Data unseen during training and validation.
- Has no influence on when to stop training.
- With early stopping, we’ve maximized the ability to generalize **to the validation set**;
- To judge the final result, we should measure its ability to generalize to completely unseen data.

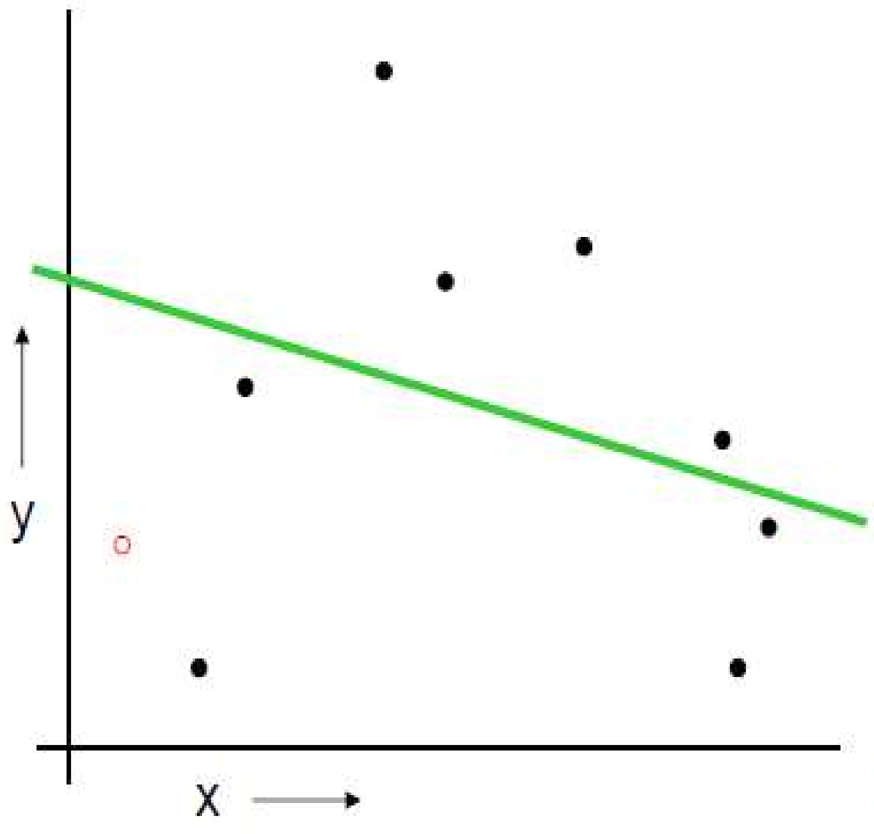
k-Fold Cross Validation

- Validation and testing leaves less training data.
- Solution: repeat over many different splits.



Source: Wikimedia Commons

Leave-one-out Cross Validation



For k=1 to R

1. Let (x_k,y_k) be the k^{th} record
2. Temporarily remove (x_k,y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k,y_k)

When you've done all points, report the mean error.