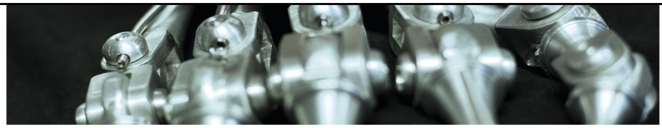


UiO : Department of Informatics
University of Oslo

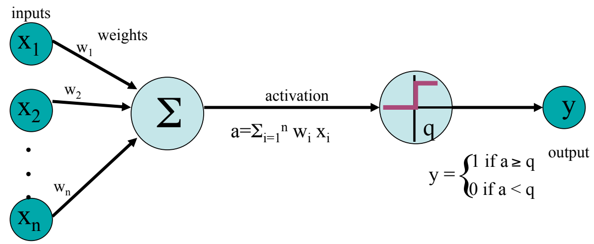
INF3490 - Biologically inspired computing

Lecture 5th October 2015

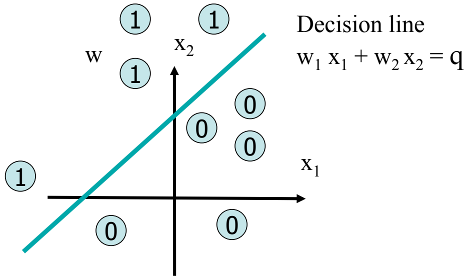
Multi-Layer Neural Network

Jim Tørresen

A Quick Overview (Perceptron)



A Quick Overview (Decision Surface)



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?***

UiO : Department of Informatics
University of Oslo

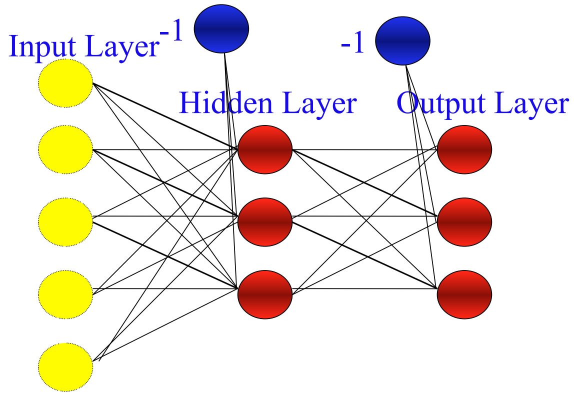
A Quick Overview

- ***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.

5

UiO : Department of Informatics
University of Oslo

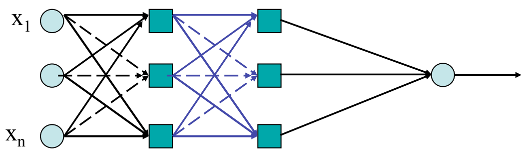
Multi-Layer Perceptron (MLP)



6

UiO : Department of Informatics
University of Oslo

1st Question?



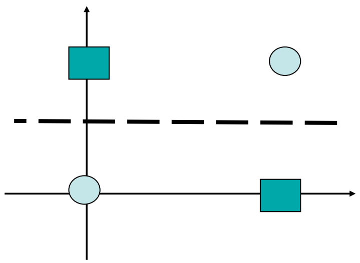
What do the extra layers gain you?

Start with looking at what a single layer can't do.

7

UiO : Department of Informatics
University of Oslo

XOR Problem



XOR (Exclusive OR) Problem

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

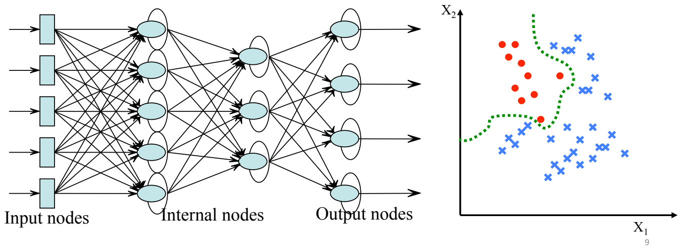
Perceptron does not work here.

Single layer generates a linear decision boundary.

8

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.



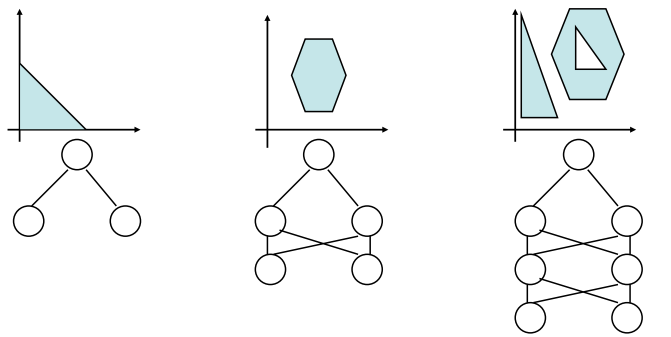
Multilayer Network Structure

- 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 Network

- Layer *n-1* is fully connected to layer *n*.
- 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.

What Do Each of The Layers Do?



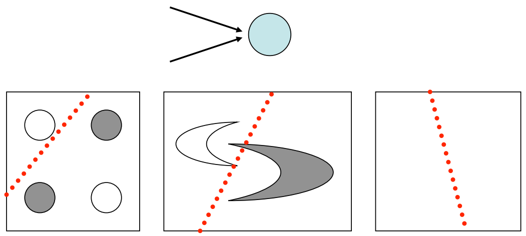
1st layer draws linear boundaries

2nd layer combines the boundaries

3rd layer can generate arbitrarily complex boundaries

UiO : Department of Informatics
University of Oslo

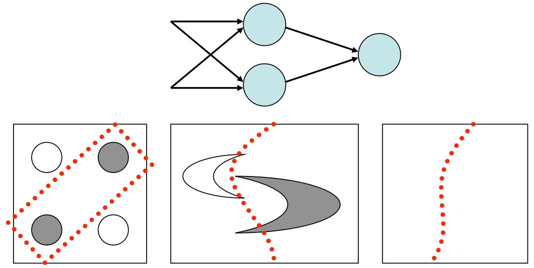
MultiLayer Perceptron: Decision Boundaries



Straight lines (surfaces), linear separable, half plane bounded by hyperplane.

UiO : Department of Informatics
University of Oslo

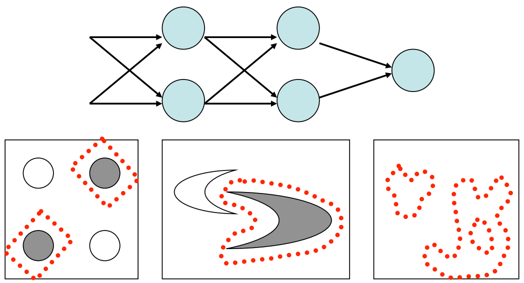
Multi Layer Perceptron: Decision Boundaries



Convex areas (open or closed).

UiO : Department of Informatics
University of Oslo

MultiLayer Perceptron: Decision Boundaries

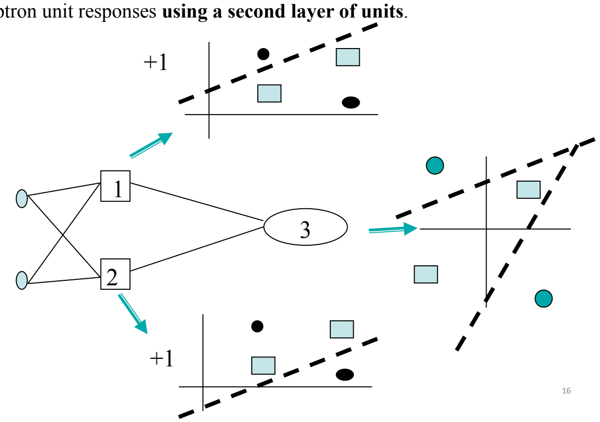


Combinations of convex areas.

UiO : Department of Informatics
University of Oslo

Solution for XOR : Add a Hidden Layer !!

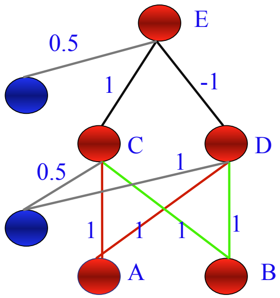
Minsky & Papert (1969) offered **solution to XOR problem** by combining perceptron unit responses **using a second layer of units**.



UiO : Department of Informatics
University of Oslo

XOR Again

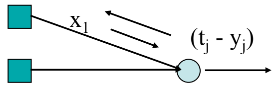
A	B	C_{in}	C_{out}	D_{in}	D_{out}	E_{in}
0	0	-0.5	0	-1	0	-0.5
0	1	0.5	1	0	0	0.5
1	0	0.5	1	0	0	0.5
1	1	1.5	1	1	1	-0.5



UiO : Department of Informatics
University of Oslo

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.

UiO : Department of Informatics
University of Oslo

Then…

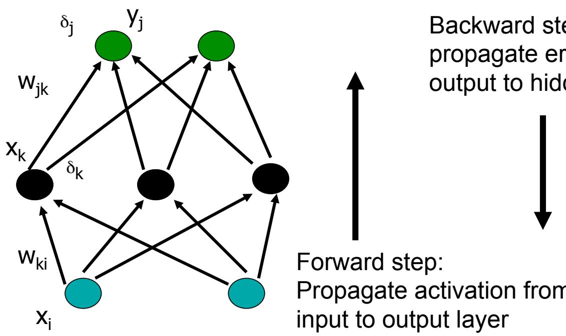
The problem is: ***How to learn Multi Layer Perceptrons??***

Solution: Backpropagation Algorithm (Rumelhart and colleagues,1986)

UiO : Department of Informatics
University of Oslo

Backpropagation

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



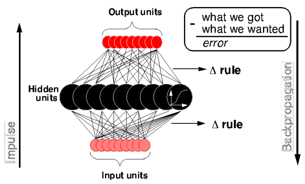
Backward step: propagate errors from output to hidden layer

Forward step: Propagate activation from input to output layer

UiO : Department of Informatics
University of Oslo

Backpropagation of Error

- During the backward pass the weights are adjusted in accordance with the ***error correction*** *rule*.
- The error is the ***actual*** output is subtracted from the ***desired*** output.
- The weights are adjusted to minimize this error.



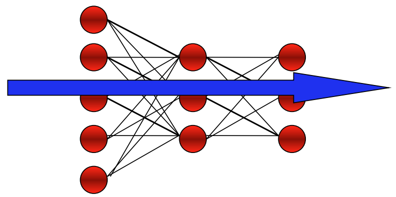
Neural Networks and Logistic Regression by Lucila Ohno-Machado Decision Systems Group, Brigham and Women's Hospital, Department of Radiology

UiO : Department of Informatics
University of Oslo

Training MLPs

Forward Pass

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.

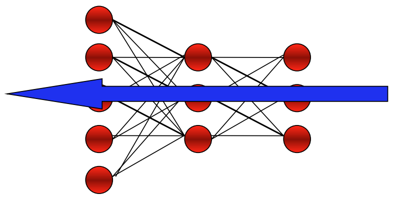


UiO : Department of Informatics
University of Oslo

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.



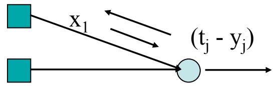
UiO : Department of Informatics
University of Oslo

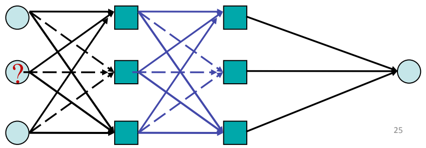
Back Propagation Algorithm

- 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 weights ***converge*** and the error is reduced to an acceptable value.

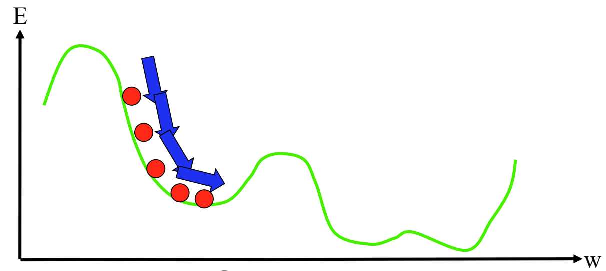
Gradient Descent Learning

- Target: Minimize the error.
- Harder than Perceptron:
  - Many weights
  - Which ones are wrong; input-hidden or hidden-output?
- Use **gradient descent learning**
- Compute gradient => differentiate sum-of squares error function.





Gradient Descent



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

← The weight is the only factor relevant to the error.

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***:

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

Error Terms

- Need to differentiate the **activation function**
- ***Chain rule of differentiation***.
- Gives us the following *error terms* (deltas)
  - For the outputs

$$\delta_k = (y_k - t_k)\, y_k (1 - y_k)$$

  - For the hidden nodes

$$\delta_j = a_j(1 - a_j) \sum_k w_{jk} \delta_k$$

UiO : Department of Informatics
University of Oslo

Update Rules

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

$$w_{jk} \leftarrow w_{jk} - \eta \delta_k a_j^{\text{hidden}}$$

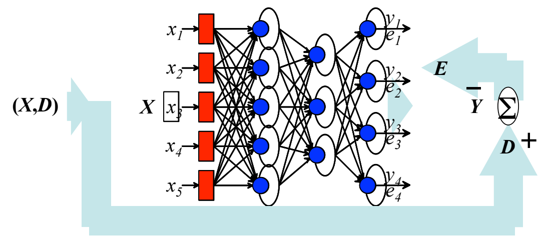
  - For the weights on the hidden nodes:

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

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

UiO : Department of Informatics
University of Oslo

BackPropagation Algorithm



UiO : Department of Informatics
University of Oslo

Summary of Backpropagation

1. Introduce inputs.
2. Feed values forward through network.
3. Compute sum-of-squares error at outputs.
4. Compute the delta terms at the output by differentiation.
5. Use this to update the weights connecting the last hidden layer to the outputs.
6. Once these are correct, propagate deltas back to the neurons of the hidden layers.
7. Compute the delta terms for these neurons.
8. Use them to update the next set of weights.
9. Repeat until the inputs are reached.

UiO : Department of Informatics
University of Oslo

Algorithm (sequential)

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

$$\Delta_i = (d_i - y_i) g'(a_i)$$

3. Propagate deltas backwards to hidden layer deltas:

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

4. Update weights:

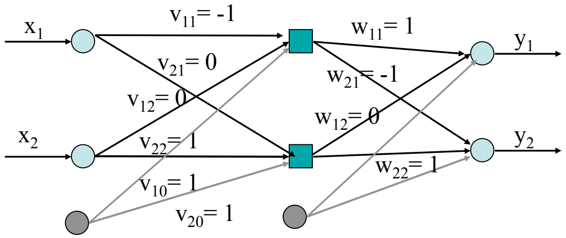
$$v_{ij} \leftarrow v_{ij} + \eta \delta_i x_j$$

$$w_{ij} \leftarrow w_{ij} + \eta \Delta_i z_j$$

UiO : Department of Informatics
University of Oslo

Example: Backpropagation

Once weight changes are computed for all units, weights are updated at the same time (bias included as weights here). An example:



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

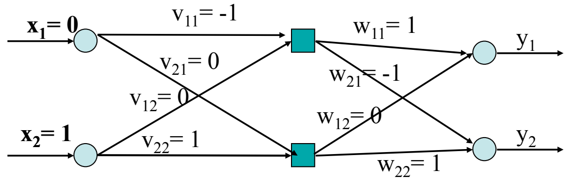
33

UiO : Department of Informatics
University of Oslo

Example: Backpropagation

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

Learning rate h = 0.1



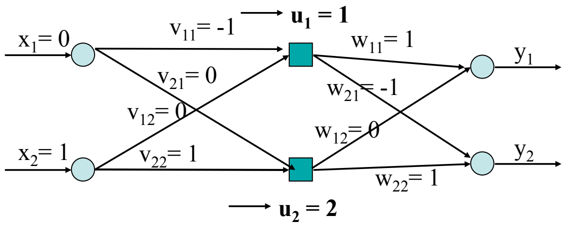
Have input [0 1] with target [1 0].

34

UiO : Department of Informatics
University of Oslo

Example: Backpropagation

Forward pass. Calculate 1st layer activations:



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

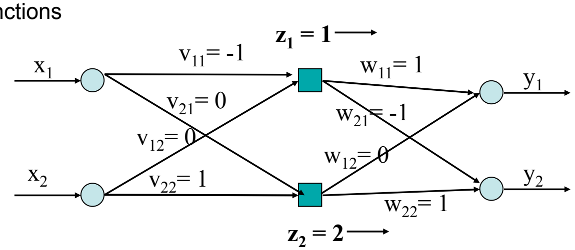
$u_2 = 0x0 + 1x1 + 1 = 2$

35

UiO : Department of Informatics
University of Oslo

Example: Backpropagation

Calculate first layer outputs by passing activations thru activation functions



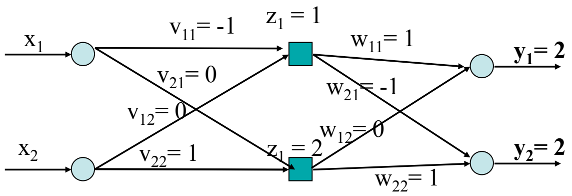
$z_1 = g(u_1) = 1$

$z_2 = g(u_2) = 2$

36

Example: Backpropagation

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



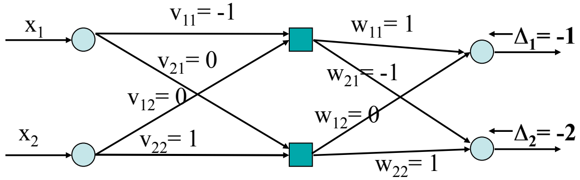
$y_1 = a_1 = 1x1 + 0x2 + 1 = 2$

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

Example: Backpropagation

Backward pass:

$$\Delta_i = (d_i - y_i)g'(a_i)$$



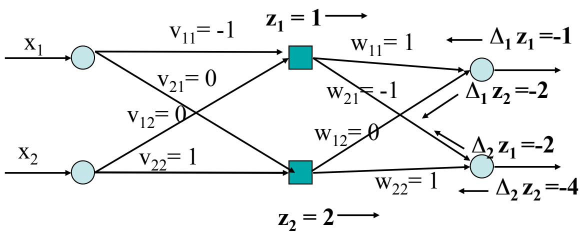
Target =[1, 0] so $d_1 = 1$ and $d_2 = 0$. So:

$\Delta_1 = (d_1 - y_1) = 1 - 2 = -1$

$\Delta_2 = (d_2 - y_2) = 0 - 2 = -2$

Example: Backpropagation

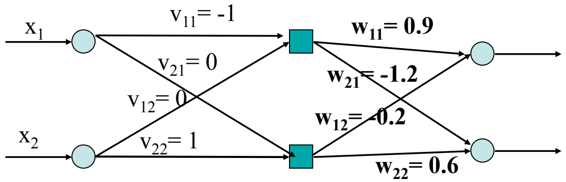
Calculate weight changes for 1[st] layer (cf perceptron learning):



$$w_{ij} \leftarrow w_{ij} + \eta \Delta_i z_j$$

Example: Backpropagation

Weight changes will be:

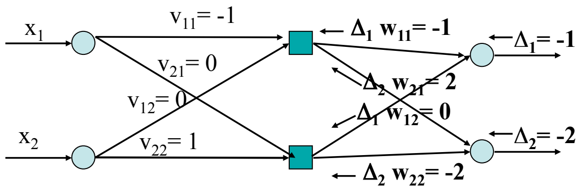


$$w_{ij} \leftarrow w_{ij} + \eta \Delta_i z_j$$

UiO : Department of Informatics
University of Oslo

Example: Backpropagation

Calculate hidden layer deltas:



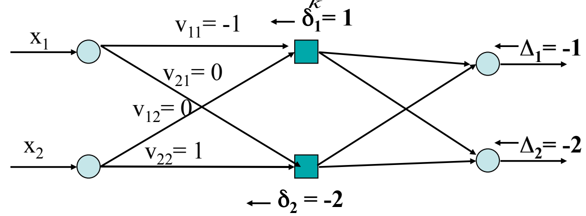
$$\delta_i = g'(u_i)\sum_k \Delta_k w_{ki}$$

UiO : Department of Informatics
University of Oslo

Example: Backpropagation

D's propagate back:

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

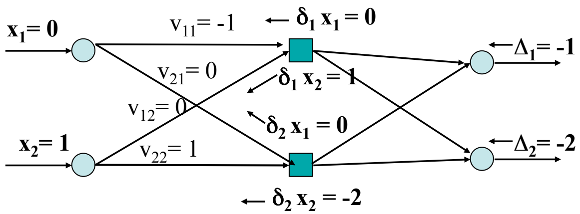


$\delta_1 = - 1 + 2 = 1$
$\delta_2 = 0 - 2 = -2$

UiO : Department of Informatics
University of Oslo

Example: Backpropagation

And are multiplied by inputs



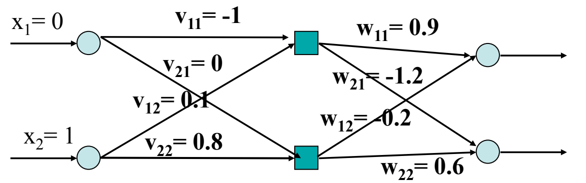
$$v_{ij} \leftarrow v_{ij} + \eta \delta_i x_j$$

UiO : Department of Informatics
University of Oslo

Example: Backpropagation

Finally change weights:

$$v_{ij} \leftarrow v_{ij} + \eta \delta_i x_j$$

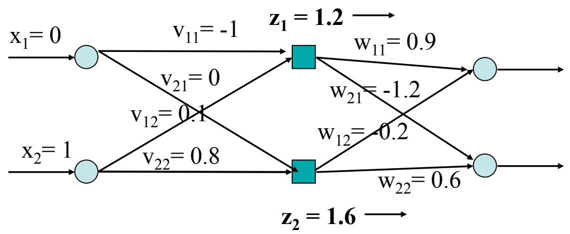


Note that the weights multiplied by the zero input are unchanged as they do not contribute to the error

We have also changed biases (not shown)

Example: Backpropagation

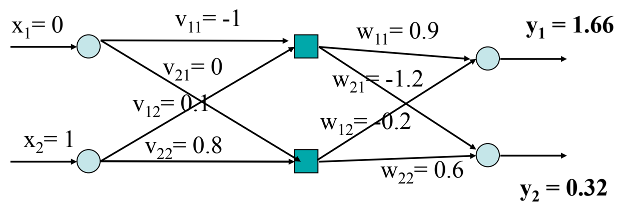
Now go forward again (would normally use a new input vector):



UiO : Department of Informatics
University of Oslo

Example: Backpropagation

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



Outputs now closer to target value [1, 0]

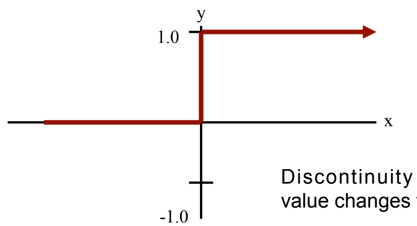
UiO : Department of Informatics
University of Oslo

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)

UiO : Department of Informatics
University of Oslo

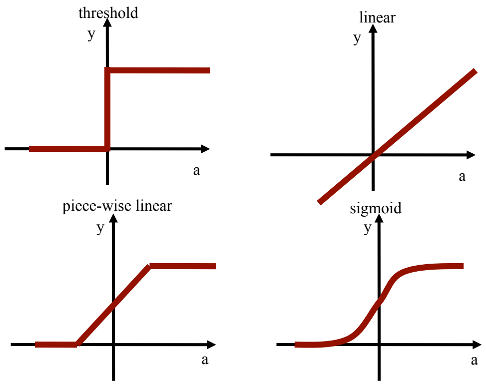
Hard Limit Function



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

UiO : **Department of Informatics**
University of Oslo

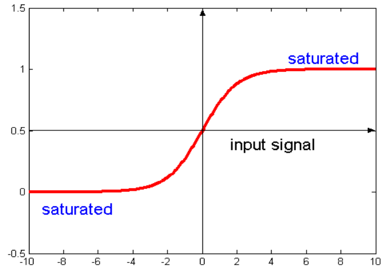
A Quick Overview (Activation Functions)



UiO : **Department of Informatics**
University of Oslo

Sigmoidal (Logistic) Function-Common in MLP

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

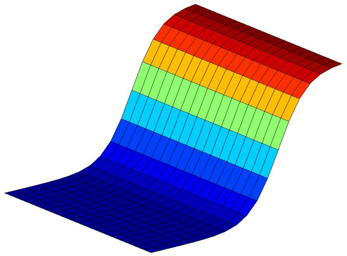


- Where *k* is a positive constant.
- The sigmoidal function gives a value in range of 0 to 1.
- Alternatively can use *tanh(ka)* which is same shape but in range –1 to 1.

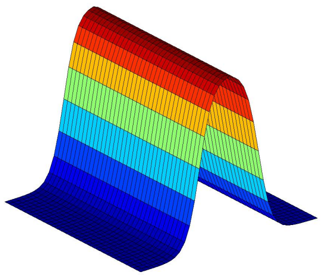
Note: when net $= 0$, $g = 0.5$

UiO : **Department of Informatics**
University of Oslo

Learning Capacity



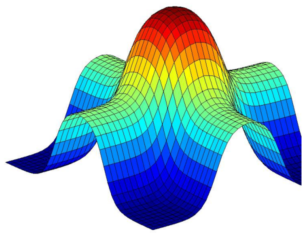
Output of one sigmoid



Addition of two sigmoids

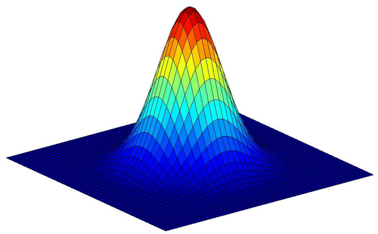
UiO : **Department of Informatics**
University of Oslo

Learning Capacity



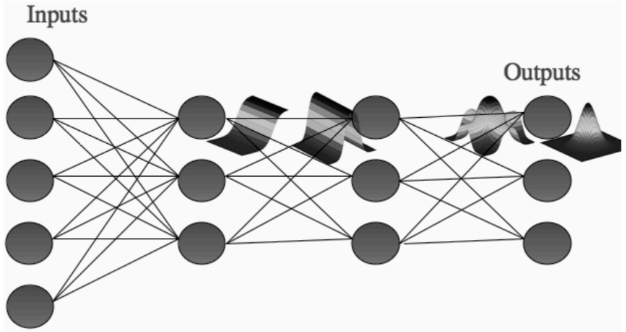
Addition of two ridges
Unique maximum

Addition of more ridges and transformation with another sigmoid
Localised response



UiO : Department of Informatics
University of Oslo

Learning Capacity



Any function can be approximated as the summation of many responses

UiO : Department of Informatics
University of Oslo

Selecting Initial Weight Values

- The MLP algorithm suggest that weights are initialized to **small random numbers** (<± 1), both positive and negative
- Choice of initial weight values is important as this decides starting position in weight space. That is, how far away from global minimum
- Aim is to select weight values which produce midrange function signals (not in only saturated signal, see sigmoid function)
- Select weight values randomly from uniform probability distribution
- Normalise weight values so number of weighted connections per unit produces midrange function signal

UiO : Department of Informatics
University of Oslo

Network Training

- Training set shown repeatedly until stopping criteria are met.
- Usual to randomize order of training patterns presented for each epoch in order to avoid correlation between consecutive training pairs being learnt (order effects).
- ***When should the weights be updated?***
  - After all inputs seen (***batch***)
    - More accurate estimate of gradient
    - Converges to local minimum faster (Jim doesn´t agree!)
  - After each input is seen (***sequential***)
    - Simpler to program and most commonly used
    - May escape from local minima (change order or presentation)
- Both ways, **need many epochs** - passes through the whole dataset

UiO : Department of Informatics
University of Oslo

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.

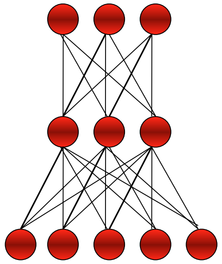
Input Normalization

- Stops the weights from getting unnecessarily large.
- Treat each data dimension independently.
- Each input variable should be processed so that the mean value is close to zero or at least very small when compared to the standard deviation.

UiO : Department of Informatics
University of Oslo

Amount of Training

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



UiO : Department of Informatics
University of Oslo

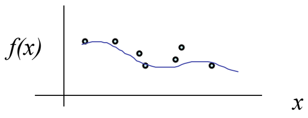
Generalisation

- 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; otherwise we would just use a look-up table.
- Under-training is ***bad***.
- Over-training is also ***bad.***

UiO : Department of Informatics
University of Oslo

Generalization

- Generalization can be viewed as a mathematical ***interpolation*** or ***regression*** over a set of training points:



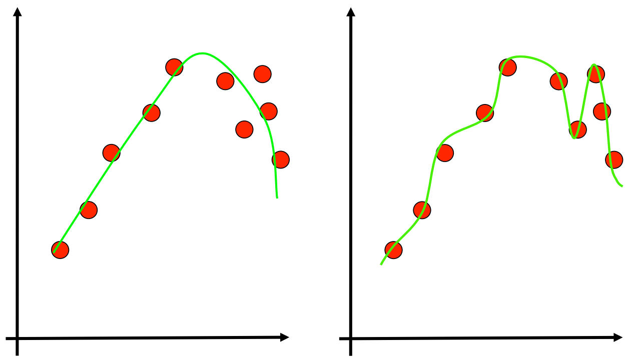
UiO : Department of Informatics
University of Oslo

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.

UiO : Department of Informatics
University of Oslo

Overfitting



UiO : Department of Informatics
University of Oslo

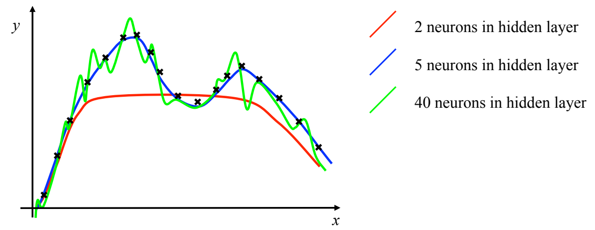
Overfitting

- 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***.

UiO : Department of Informatics
University of Oslo

The Problem of Overfitting

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



UiO : Department of Informatics
University of Oslo

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.

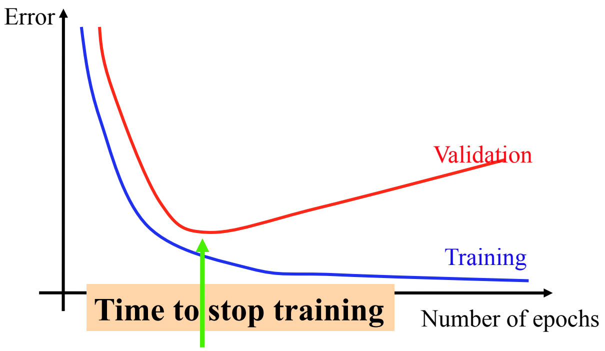
UiO : Department of Informatics
University of Oslo

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.

UiO : Department of Informatics
University of Oslo

Early Stopping



UiO : Department of Informatics
University of Oslo

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.

UiO : Department of Informatics
University of Oslo

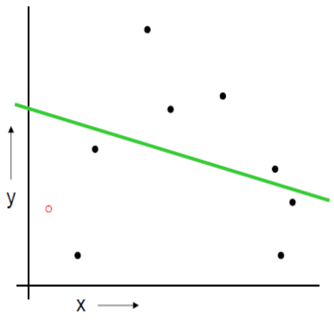
k-Fold Cross Validation

- Cross-validation leaves less training data.
- Generalization ability is still only measured on a small set (which will be biased).
- Solution: repeat over many different splits.
  - Divide all data into *k* sets (or folds).
  - For i = 1…*k*:
    - Train on data[i], validate on data[i+1], test on rest.
  - Average the results.

69

UiO : Department of Informatics
University of Oslo

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.

70

UiO : Department of Informatics
University of Oslo

Some questions

- What is overfitting?
- How do we avoid overfitting?
- What do you do if you have limited data and would like to do validation?

71