SNNS ANALYZE

- Program

  Directory:
  /KDrive/SEH/SCSIT/Students/Courses/COSC2111/DataMining/
  Programs:
  javanns/analyse (Titan/Saturn/Jupiter)
  javanns/analyse.exe (PC)

- Documentation

http://www.ra.cs.uni-tuebingen.de/SNNS/UserManual/node324.htm
/KDrive/SEH/SCSIT/Students/Courses/COSC2111/DataMining/
snns/SNNSv4.2.Manual.pdf [Chap 13]

JavaNNS Training with a validation file

1. How to train with a validation file
2. How to get a result file
3. How to get the test error rate

JavaNNS Result File

```
SNNS result file V1.4-3D
generated at Wed Apr 18 09:45:33 2012

No. of patterns     : 100
No. of input units  : 4
No. of output units : 3
startpattern        : 1
endpattern          : 100
input patterns included
teaching output included
#1.1
4.4 3 1.3 0.2
1 0 0
0.9005 0.22446 0
#2.1
4.3 3 1.1 0.1
1 0 0
0.90062 0.22446 0
#3.1
6 3.4 4.5 1.6
0 1 0
0 0.32981 0.65058
.......
......
STATISTICS ( 100 patterns )
wrong   : 13.00 %  ( 13 pattern(s) )
right   : 68.00 %  ( 68 pattern(s) )
unknown : 19.00 %  ( 19 pattern(s) )
error   : 31.074306
```

Titan/Saturn/jupiter command:

/KDrive/SEH/SCSIT/Students/Courses/COSC2111/DataMining/
javanns/analyze -s -i iris.res

JavaNNS CREATE NEW NETWORK

1. Tools
2. First create layers
3. Then create connections

CLASSIFICATION WITH ANNS: ERROR

Using ANN for a classification problem we need to distinguish

1. TSS (or MSE) on Training Set

$$\frac{1}{2}\sum_{patterns}\sum_{z}(d_z - o_z)^2 \tag{1}$$

   z output units in network
   d_z is desired output for node z
   o_z is actual output for node z

2. TSS (or MSE) on Test Set

3. Classification Error on Training Set

4. Classification Error on Test Set

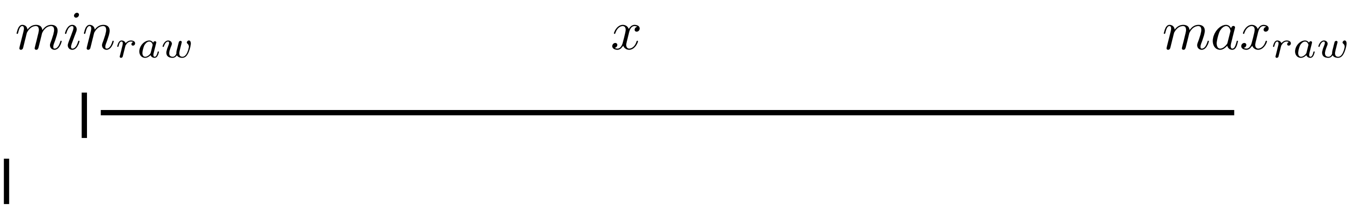
   Apply 402040 or 500050 to test data
   Count number correct, incorrect, unclassified

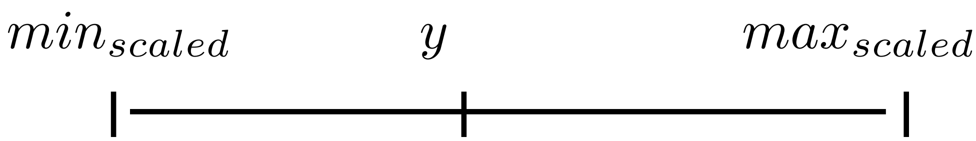
NUMERIC PREDICTION WITH ANNS

```
@relation 'bodyfat.names'
@attribute Density real
@attribute Age real
@attribute Weight real
@attribute Height real
@attribute Neck_Circumference_CM real
@attribute Chest_Circumference_CM real
@attribute Abdomen_Circumference_CM real
@attribute Hip_Circumference_CM real
@attribute Thigh_Circumference_CM real
@attribute Knee_Circumference_CM real
@attribute Ankle_Circumference_CM real
@attribute Biceps_Circumference_CM real
@attribute Forearm_Circumference_CM real
@attribute Wrist_Circumference_CM real
@attribute class_Percent_Bodyfat real
@data
1.0708,23,154.25,67.75,36.2,93.1,85.2,94.5,59,37.3,21.9,32,27.4,17.1,12.3
1.0853,22,173.25,72.25,38.5,93.6,83,98.7,58.7,37.3,23.4,30.5,28.9,18.2,6.1
```

- Network Architecture: 14-h-1
- Create new training (and test) set in which class variable is scaled to range $[0, 1]$
- [Note: A Weka filter can do it.]
- Train network
- Apply to test data
- Reverse the scaling for predictions

SCALING OF VARIABLES





$$\frac{x - min_{raw}}{max_{raw} - min_{raw}} = \frac{y - min_{scaled}}{max_{scaled} - min_{scaled}}$$

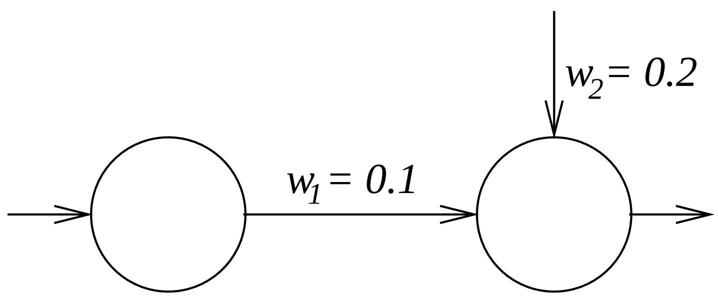
$$y = min_{scaled} + \frac{(x - min_{raw})(max_{scaled} - min_{scaled})}{max_{raw} - min_{raw}}$$

WHERE DO WEIGHTS COME FROM

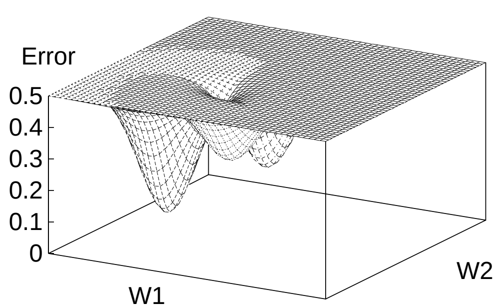
- Massively difficult problem, in general
- Much current research
- General Approach
  1. Get examples for which desired behaviour is known
  2. Pick a random set of small weights
  3. Put examples through the network giving network outputs. Difference between network outputs and desired outputs is the error
     One pass through examples during training is an Epoch (Cycle in JavaNNS)
  4. If error is small enough stop
  5. Adjust current weights to, hopefully, make error smaller
  6. Go to 3

WEIGHT OPTIMIZATION

- In general: $Error = f(w_{ij})$
- We want to find the minimum value of $Error$
- We have a multidimensional optimization problem, that is, which values for w_{ij} will give the lowest error
- The following network has 2 weights to be found.



For this network we might have an error surface like:



WEIGHT OPTIMIZATION

- The error surface was found by:
  [Kind of exhaustive search]
  For i from -20 to +20 step 0.1
    For j from -20 to +20 step 0.1
      Calculate and plot error

- The best weights are the ones corresponding to the deepest valley

- We can only do this because we have just 2 weights

- For n weights we must find the deepest valley in n dimensional space. We can't actually compute the error surface.

- Usually can't use analytic methods (Calculus)

- Need to SEARCH

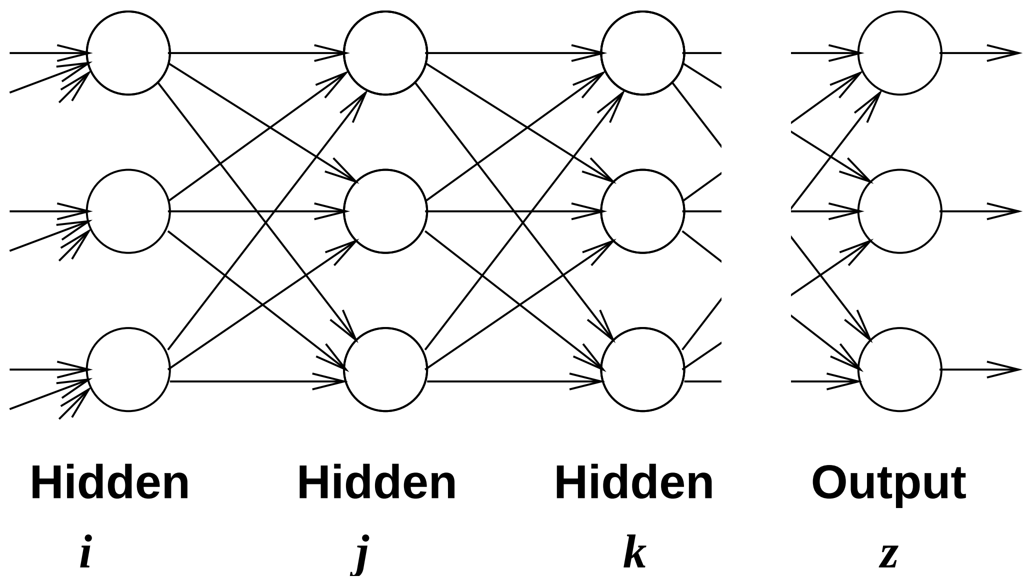
- E.g. Hill Climbing

- Just about any search method you can think of has been tried.

GRADIENT DESCENT
LOCAL MINIMA PROBLEM

- Gradient Descent
  1. Pick a point at random
  2. Look around for a lower point with distance d
  3. If not found stop
  4. Go to new point
  5. Go to 2
- Using gradient descent, some initial points will get stuck in valley that is not the deepest one.
- Partial solution is to restart with a different initial point.

INTUITION BEHIND BEP 1

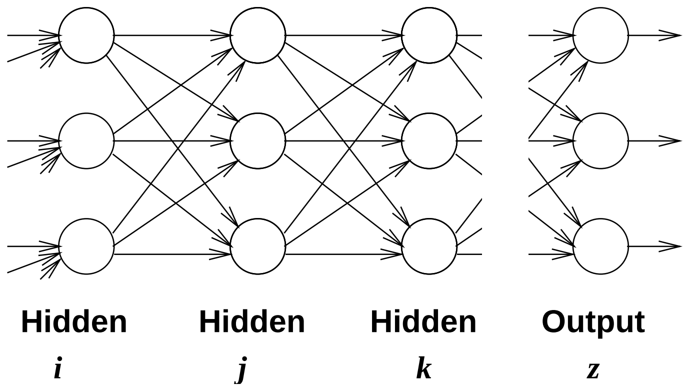


- How big a change should we make to weight $w_{i \to j}$?

- Make a big change if it will result in a big improvement in error

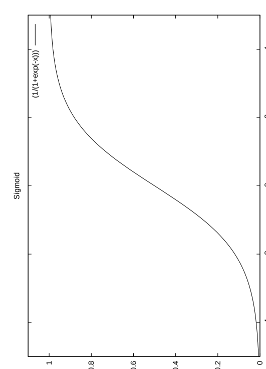
- If a change to $w_{i \to j}$ will have little effect on on error, make it small

INTUITION BEHIND BEP 2



- A change in input to node j results in a change to output that depends on the *slope* of transfer function

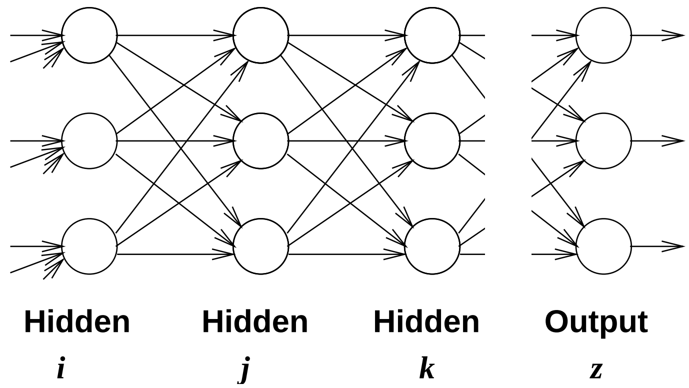
- Change in input has maximum effect where the slope is steepest



- Slope of sigmoid/logistic is given by $o(1-o)$

  Thus $\Delta w_{i \to j} \propto o_j(1 - o_j)$

INTUITION BEHIND BEP 3



- Change in input to node j depends on output of node i. $w_{i \to j}$ should change substantially if o_i is high.

  Thus $\Delta w_{i \to j} \propto o_i$

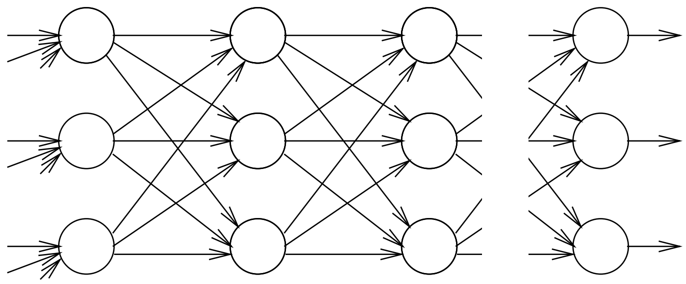
- Let β be a factor which measures how beneficial the change is (in terms of lower error). Node j is connected to nodes in next (kth) layer. A change in o_j will be a benefit to each one. So

  Hidden: $\beta_j = \sum_k w_{i \to j} o_k (1 - o_k) \beta_k$

  Output: $\beta_z = d_z - o_z$

  and $\Delta w_{i \to j} \propto \beta_j$

INTUITION BEHIND BEP 4



- Putting it all together:

  $\Delta w_{i \to j} \propto o_i o_j (1 - o_j) \beta_j$

  Let η be the constant or *learning rate*

  Back-propagation formulas

  $\Delta w_{i \to j} = \eta o_i o_j (1 - o_j) \beta_j$

  $\beta_j = \sum_k w_{i \to j} o_k (1 - o_k) \beta_k$ (Hidden units)

  $\beta_z = d_z - o_z$ (Output Units)

BASIC BACKWARD ERROR PROPAGATION

- Let η be the learning rate.
- Set all weights, including biases to small random values.
- Until total error (TSS or RMSE) is small enough do
  - For each input vector
    * Feed forward pass to get outputs
    * Compute β for output nodes $\beta_z = d_z - o_z$
    * Compute β for hidden nodes, working from last layer to first layer

      $\beta_j = \sum_k w_{i\to j} o_k (1 - o_k)\beta_k$
    * Compute and store weight changes for all weights

      $\Delta w_{i\to j} = \eta o_i o_j (1 - o_j)\beta_j$
  - Add up weight changes for all input vectors and change the weights

NOTES ON BEP

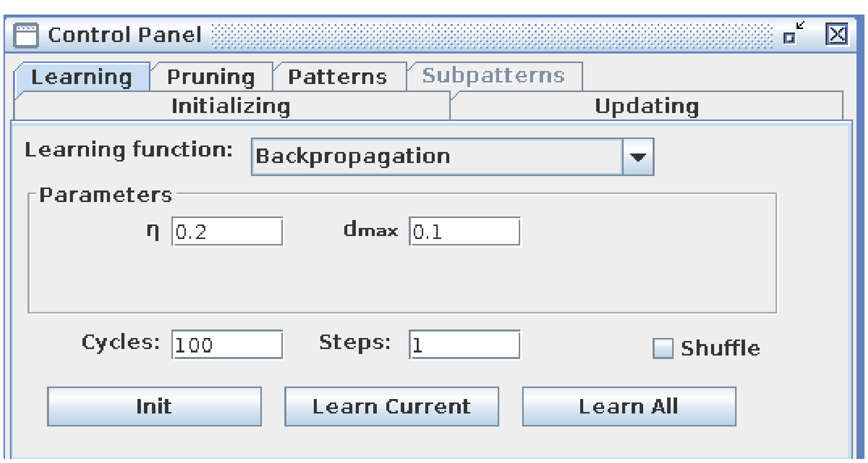
- *Epoch*: Presentation of all input vectors, calculation and carrying out of weight updates. [JavaSNNS = Cycle]

- Usually weights are updated at end of an epoch, not after each input vector

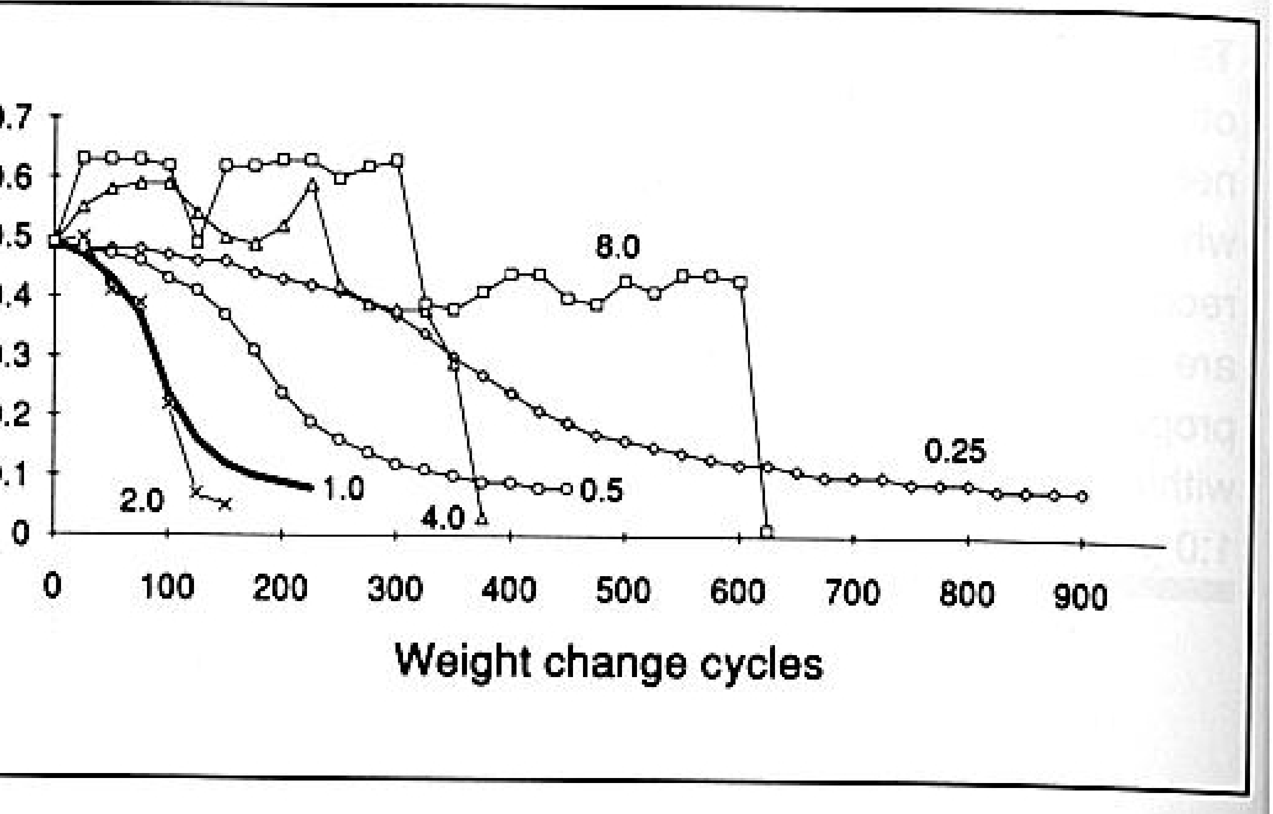
- A target of 0 or 1 can never be reached. Usually interpret a number > 0.9 or > 0.8 as 1

- Training may require thousands of epochs. A plot of TSS or RMSE during training can show how training is going

- We need to use some method for estimating the true error rate.



TE



- A learning rate of 2 appears best for this problem

- There is usually an optimal learning rate, but it is problem dependent

- In practice η = 0.2 is good choice

SPEEDING UP BEP

- It is not unusual for the training of large networks to take days or weeks

- There is a large research 'industry' looking for ways to speed up training

- Some Approaches

  - Momentum: When updating a weight, add a contribution from time $t-1$

  - (Scaled) conjugate gradient: Use the gradient from time $t-1$ and time t

  - Quickprop: Have an adaptive, dynamic momentum term

    [Compare quickprop and std-packprop on xor problem in JavaNNS]

- Unfortunately an approach will work brilliantly on one problem and be hopeless on another

- Many of these variants are implemented in JavaNNS

MOMENTUM

- Momentum: When updating a weight, add a contribution from time $t-1$

- Weight update rule becomes:

  $$\Delta w_{i\to j}(t+1) = \eta o_i o_j (1-o_j)\beta_j + \alpha \Delta w_{i\to j}(t-1)$$

- If a good direction has been found, go that way even faster

- Momentum is controvertial

  – One view: It's hopeless, don't ever use it

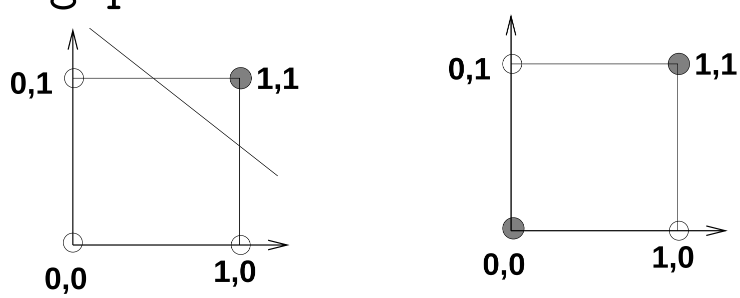
  – Opposite view: It's always better with momentum. You just have to get the right values for η and α

- Considering all pitfalls and variations, it's a wonder that any network ever gets properly trained!

WHAT CAN A NEURAL NETWORK LEARN?

- What can a two layer network (original perceptron) learn?
- It can learn to discriminate linearly separable categories such as AND.

  Note: There are many lines corresponding to different endpoints of training from differen starting points.



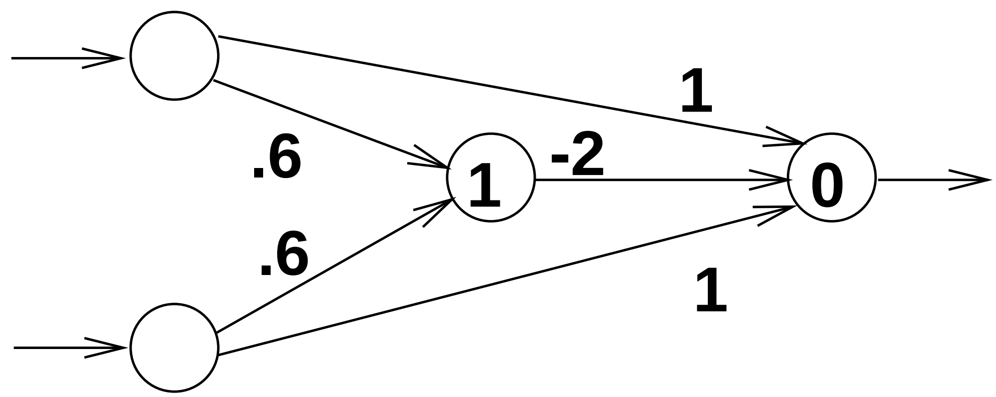
AND XOR

  XOR is linearly nonseparable. There is no way to draw a line to correctly classify all points.

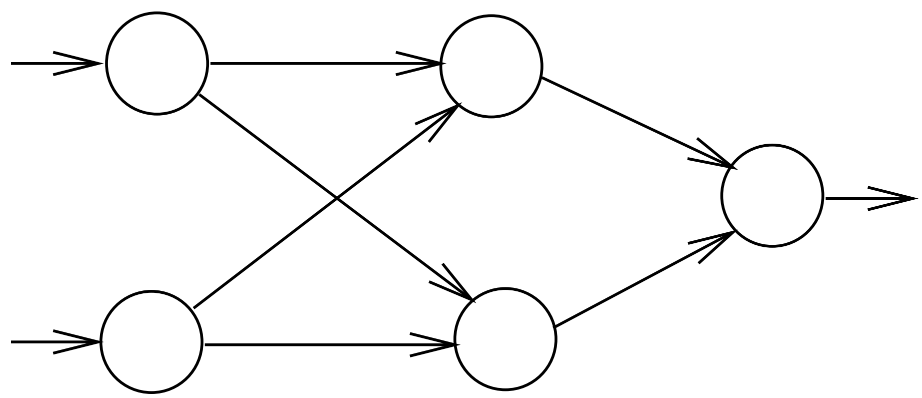
- Two layer network cannot learn the XOR function. Proved in 1969 by Minsky and Papert, with catastrophic consequences.
- Perhaps a neural network is not really useful if it can't compute something as simple as XOR?

MULTILAYER PERCEPTRONS

- Perceptron with a hidden unit and 'shortcut' connections can compute XOR



- A 3 layer network (we work with layers of units) can compute XOR.



- This was known in 1969.

- There was no generally accepted algorithm for finding the weights until mid 1980s.

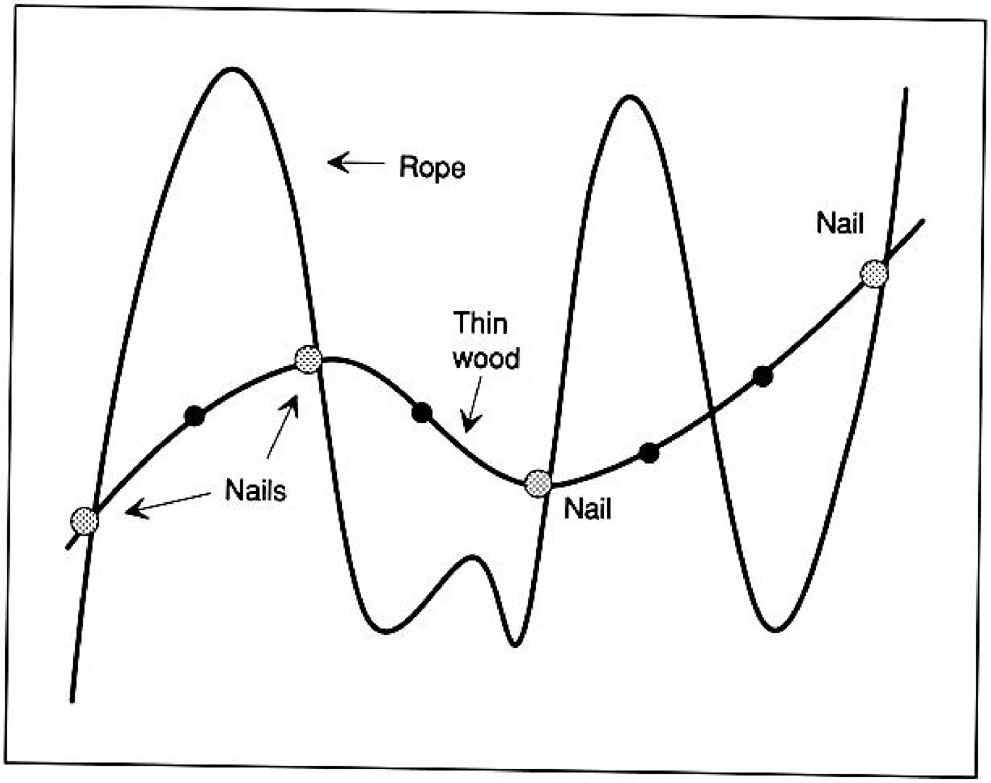
OVERFITTING

- The network has high accuracy on the data from which it was developed (training), but low accuracy on new data (test)

- Caused by
  - Training for too long
  - Having a network with too many nodes and weights
    * Each weight (and bias) is a parameter that needs to be estimated
    * The more parameters we have the more data we need for accurate estimates
    * Example: Suppose we are fitting a polynomia of degree 10 to a set of 4 points. The polynomial has 11 coefficients (parameters to be estimated). There are an infinite number of choices that give a polynomial that fits the points exactly.

      To get useful generalization the polynomial must be 'less complex' than the data itself.

OVERFITTING INTUITION



- The rope can pass through the nails in many different ways

- A flexible piece of thin wood can be stretched to fit in a few ways

- A steel rod [not shown] can only be placed in one best way.

NUMBER OF TRAINING EXAMPLES

- Heuristic: There should be at least as many training examples as weights

- Another Heuristic: There should be at least 10 times as many training examples as weights

- But some interesting work is being done with networks that have many more weights than training examples

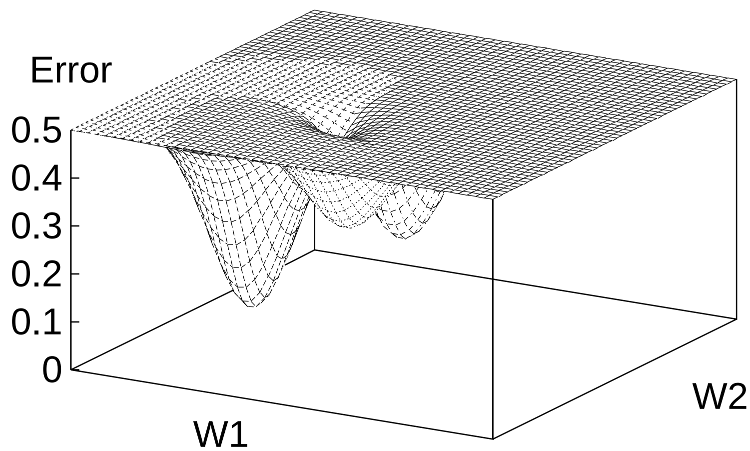
- There is currently a lot of research activity addressing the general questions of:
  - How many examples do you need to learn properly
  - Given that we have n examples, how well can we expect to learn

WHEN TO STOP TRAINING

- Stop a fixed number of epochs
- Stop when the TSS or RMSE falls below some theshold
- When error on validation set is a minimum
  - Break the TRAINING set into 2 parts
  - Use part 1 to compute the weight changes
  - Every m (Typical values 10,50,100) epochs apply the partially trained network to part 2 (the validation set) and save the weights
  - What is the expected behaviour of validation set error with training epochs?

LOCAL MINIMA

- Suppose we are training a network with two weights w_1, w_2
- For any given values of w_1 and w_2 we can work out the error. [How?]
- Plotting error vs weights might give the following error surface/landscape



- Note in general we can't do this because there are too many dimensions
- What we desire is a trajectory of points which leads us to the global minimum
- A bad trajectory or starting point will take us to a local minimum
- Oscillating trajectory

LOCAL MINIMA

- How can you tell if a local minimum has been reached?
  - A number of runs with different starting points end with (very) different TSSE
- What to do about a local minimum?
  - Nothing, if training is ‘good enough’
  - Increase the value of the learning rate η
- Oscillation
  - The TSSE graph is very ’jerky’ without an overall downward trend.
  - Decrease the learning rate.
- Sometimes decreasing the learning rate as training proceeds works well.
- We begin to see why neural net training is an art rather than a science.

NETWORK SIZE

- Usually the number of inputs and outputs is determined by the problem
- How many hidden layers? Hidden Units?
- Theorem: One hidden layer is enough for any problem
- But, training might be faster with several layers
- Best is to have as few hidden layers/nodes as possible
  - Forces better generalization
  - Fewer weights to be found
- Determining the number of hidden layers/units
  - Make the best guess you can
  - Heuristic: (attribs + classes)/2
  - Heuristic: Half the number of inputs
  - If training is unsuccessful try more hidden nodes
  - If training is successful try fewer hidden nodes
  - Inspect weights after training. Nodes which have small weights can probably be eliminated

NUMBER OF HIDDEN NODES

- In practice neural models are quite robust
- Use 3 layer networks
- Usually there is not much difference over a range of numbers of hidden nodes

REPRESENTING VARIABLES

- Two kinds of variables
  - numeric (continuous, quantitative)

    A quantity is measured on some scale: Age, weight, temperature
  - nominal (symbolic, class, non-numeric)

    Variable denotes a class (category, property, action)

    sex = male, female
    colour = blue, red, green
    action = accept, rework, discard

- Note the mapping

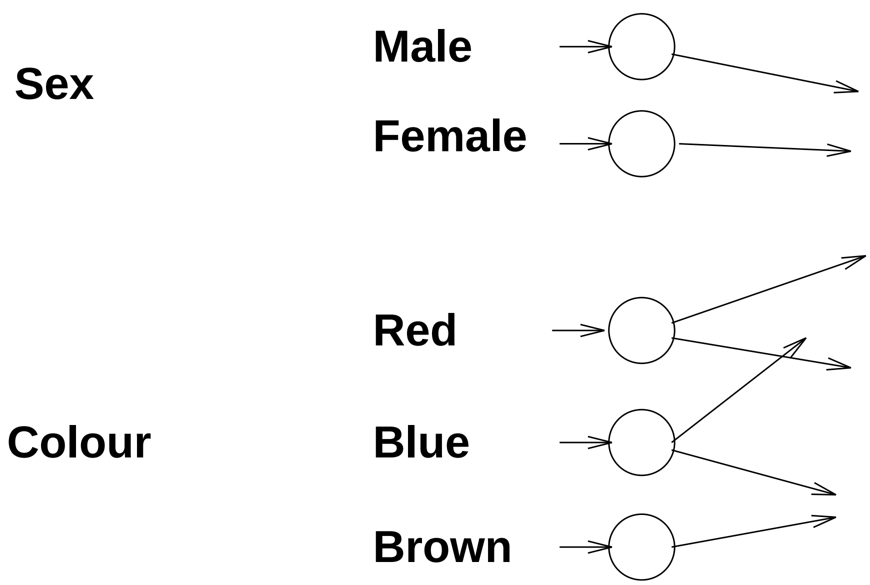
$$\begin{array}{lcl} \texttt{accept} & = & 1 \\ \texttt{rework} & = & 2 \\ \texttt{discard} & = & 3 \end{array}$$

does NOT!! turn this variable from a class variable to a numeric variable.

NOMINAL/CLASS INPUT VARIABLES

- Use a binary representation, one input node for each class.

  1 means in the class (has property), 0 means not in the class (does not have property)



Male	=	1,0
Female	=	0,1
Missing	=	0,1

Red	=	1,0,0
Blue	=	0,1,0
Brown	=	0,0,1
Missing	=	0,0,0

NOMINAL INPUT VARIABLES

- A network with above repn cannot generalize between classes since a node representing a class has no effect if not turned on.
  - Consider predicting ‘creditworthy’ based on occupation = (plumber, electrician). What the network learns about plumbers will not generalize to electricians.
  - Perhaps plumbers and electricians should be in the same class since we expect them to have same creditworthy behaviour.
- Classes must have a significant number of representatives.
  - Look for small classes that can be combined with large classes based on donain knowledge.
  - Consolidate classes based on domain knowledge. Eg put ‘thunderstorms’ with ‘heavy rain’.
- Suppose we have 10 inputs each with 10 values. How many training cases are needed to have an example of each class?
- Careful analysis and selection of inputs is necessary.

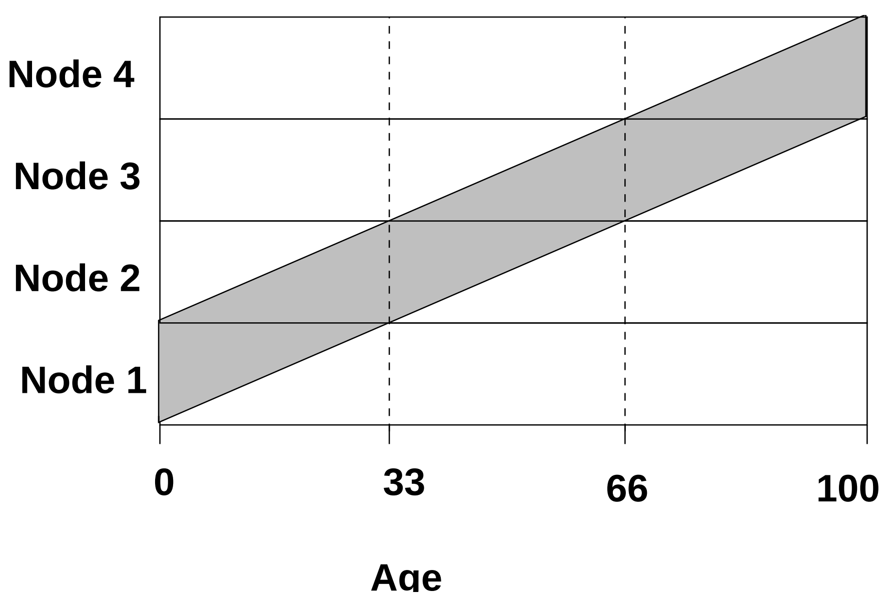
NUMERIC INPUT VARIABLES

- We can have:
  - Continuous (age, income)
  - Peridic (Wind direction, time of day, month)
  - Ordinal (First, second,.....)
- Just allocate one input node to each variable? Not always.
- Normalize inputs? Generally a good idea.

  Note Normalize $\neq$ scale
- Scale to range $[0, 1]$?
  Scale to range $[-1, 1]$?
  Scale to range $[0.1, 0.9]$?
- Opinions differ. Answer would seem to depend on the problem and the simulator.

NUMERIC ATTRIBUTES INTERPOLATION REPN

- Common to use a single node
- Multi node repn can work better
- In medical diagnosis, children, adults and old people have different medical problems. A multi node representation of age can lead to better diagnosis.



4 nodes usedfor age	0	33	66	100
A 66 year old will be	0	0	1	0
A 60 year old will be	0	.1	.9	0

- Value of a node is the fraction of its band that is grey on the vertical line corresponding to age.

PERIODIC VARIABLES

- The wind blowing from a direction of 1 degree is very close to a wind from 359 degrees

- Dec 31 (day 365) of one year should have a representation close to Jan 1 (day 1) of the next year.

- Avoid ''Representational Cliffs''

1. Use a periodic function like $sine$.

2. Use an interpolation representation.