Neural Networks Part 3

Kevin Sim

Slides courtesy of Prof. Emma Hart

RECAP

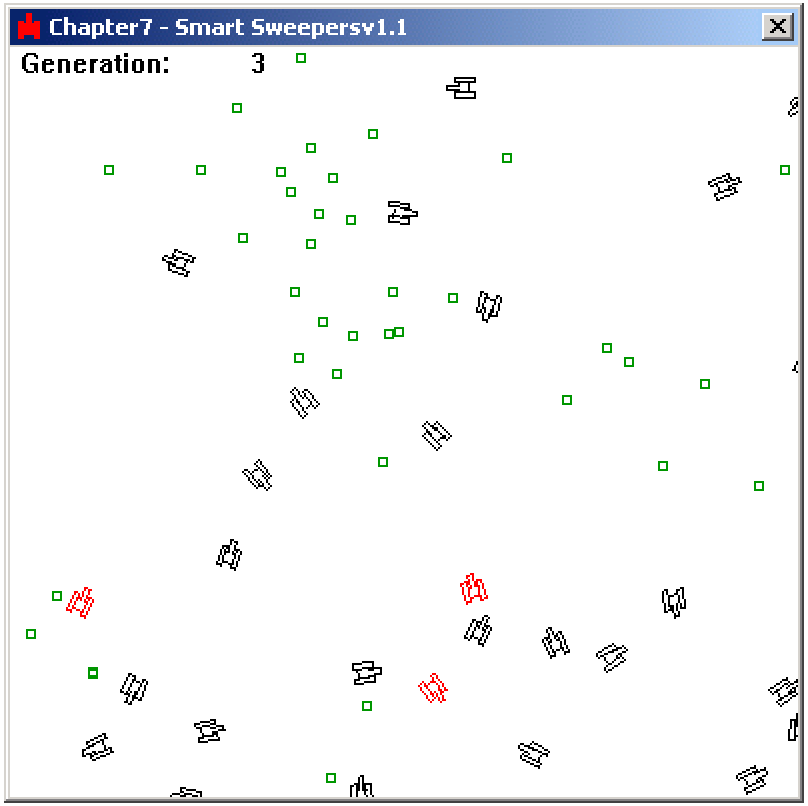
- Last week we looked at supervised learning using a multi-layer perceptron for classification
- Backpropagation
  - Finds a set of weights that gives good performance
  - Requires training data: input and output pairs
  - Iteratively reduces error at the output neurons
- For some applications:
  - Hard to generate the required training data (pairs)
  - We don’t know what the output should be

Overview

- Unsupervised learning using NN
- Training a neural network with an evolutionary algorithm
  - i.e. use EA to find a set of weights
- Applications of this:
  - Robot control with a neural network
- NEAT: evolving topology & weights of a network with an EA

An Example: Minesweeper

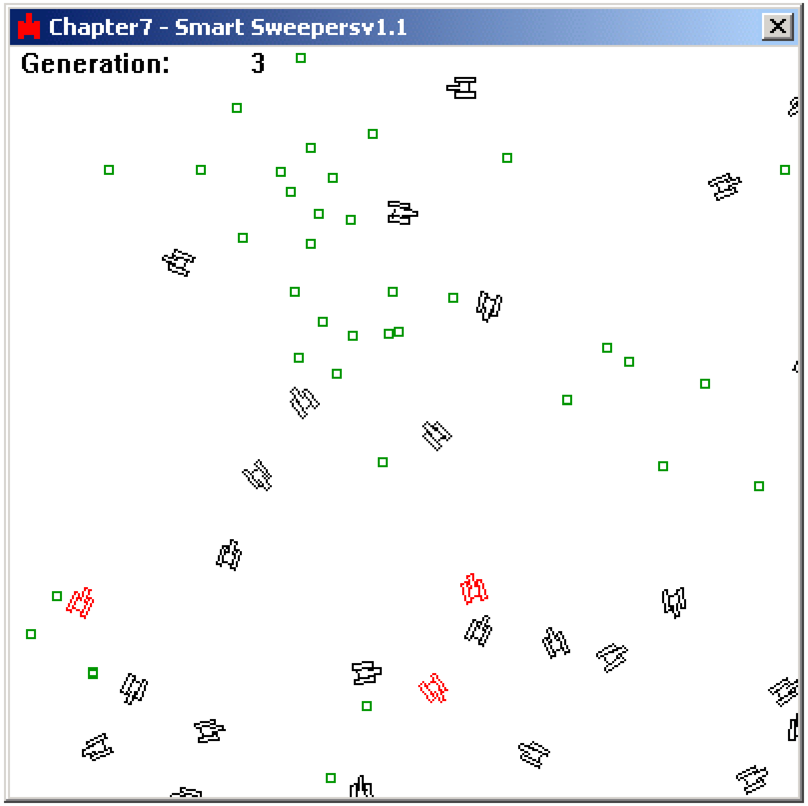
- Objective is to collect mines
- Neural network controls the movement of a “robot” (minesweeper)
- From last week we know a NN requires inputs and outputs
- Inputs are sensory information obtained from the environment
- Output is direction of movement
- NN Controller is trained to direct the minesweeper towards mines



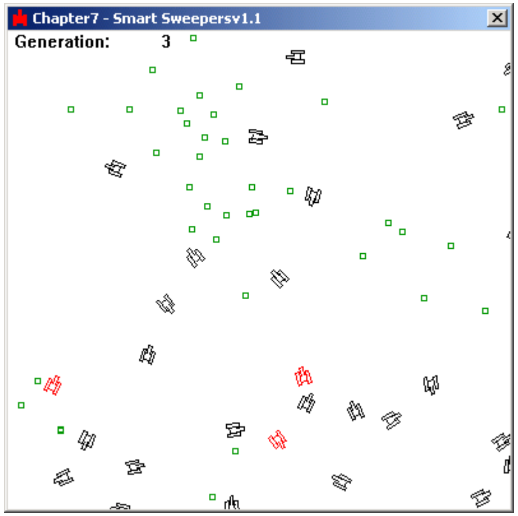
An Example: Minesweeper

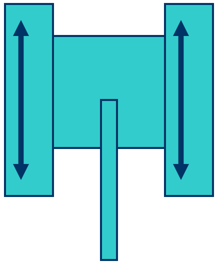
- Objective is to collect mines
- EA used to evolve weights for the NN controller
- EA Requires a representation (chromosome genotype) and fitness function (more later)
- [Example: Natural Motion](https://www.youtube.com/watch?v=ySRvKzZsDqw)

https://www.youtube.com/watch?v=ySRvKzZsDqw



Minesweeper: Inputs

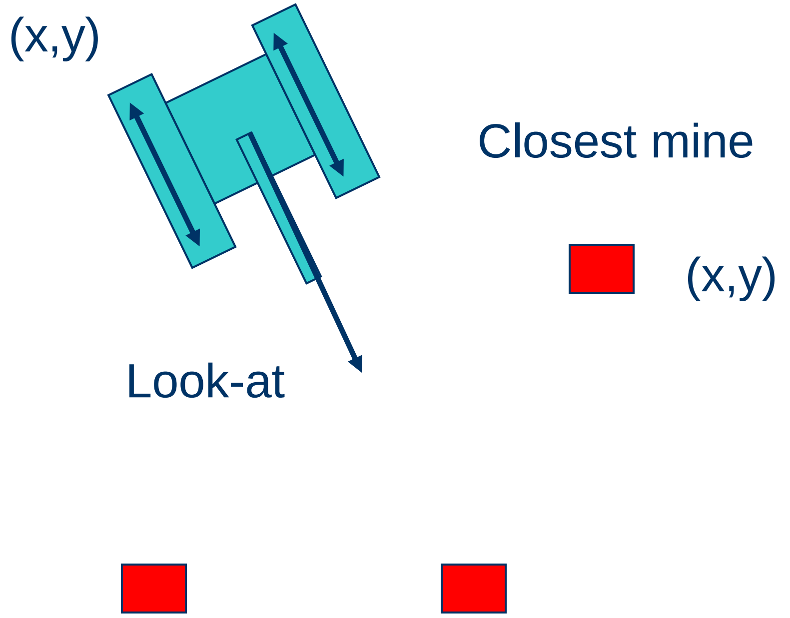




- In robotics, inputs obtained by sensing the environment
- In real-robots, could be infra-red, acoustic, video etc.
- In simulation, we can ‘sense’ distance to an obstacle

Minesweeper: Inputs

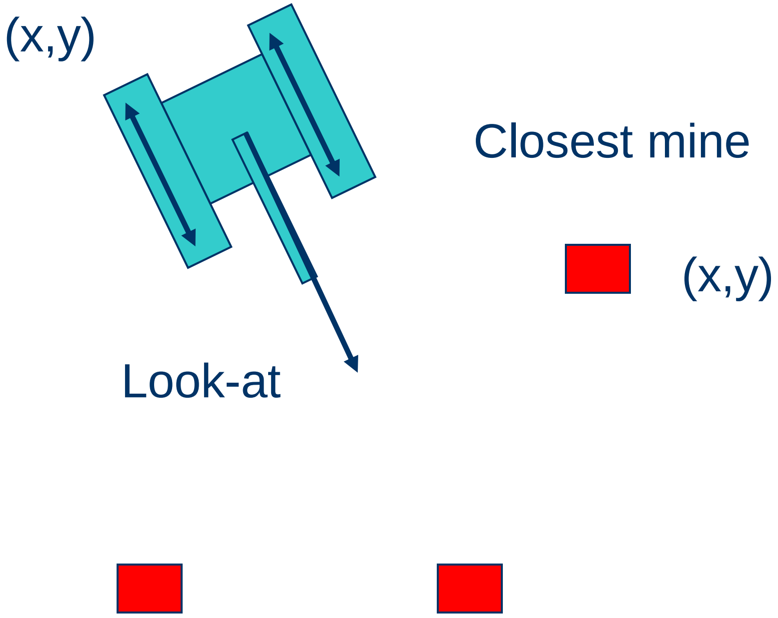
- What information does it need ?
  - Minesweepers position (x,y)
  - Position of closest mine (x,y)
  - Vector representing the heading (x,y)*
- **6 inputs in total**



Assume vectors relative to origin so defined by 2 values

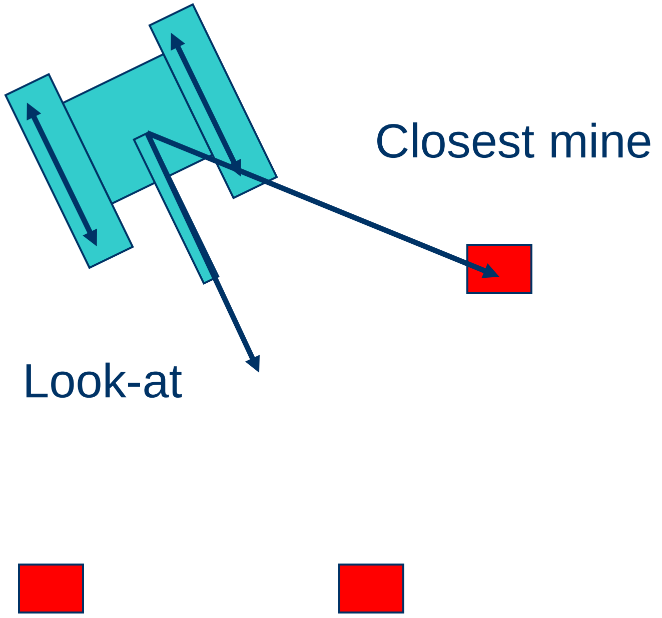
Minesweeper: Inputs

- Could use 6 inputs: But
  - Fewer inputs: fewer weights
  - Fewer weights: faster training
  - Faster training: faster network



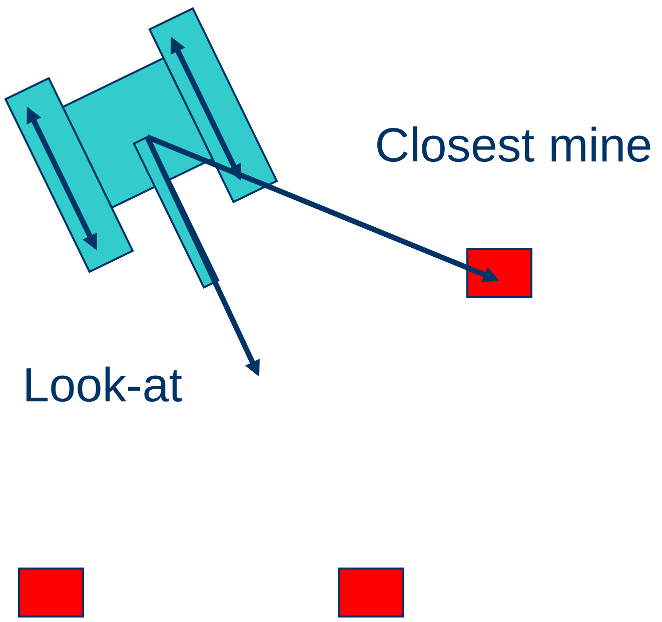
Minesweeper: Inputs

- Actual positions don't matter
- The relative direction between the mine and the current direction **does**
- Input can actually be represented by **2 vectors**
- This only requires 4 inputs:
  - (x,y) for each vector



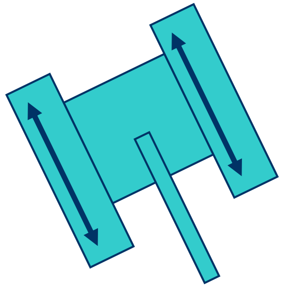
Minesweeper: Inputs

- NN require normalised data
- Good idea to standardize inputs:
  - Look-at vector is normalised to be of length 1
  - Closest-mine vector might be very large
    - We can normalize this too
- Both inputs then have similar emphasis
  - Actual distances and directions aren't important just relative difference between look at and closest mine



Minesweeper: Outputs

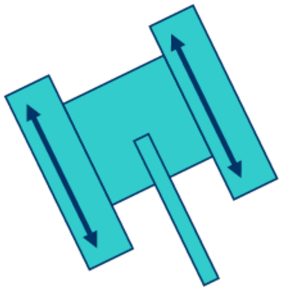
- The rotation and velocity are adjusted by activating one or both of the left and right tracks
- The NN needs two outputs:
  - Left track
  - Right track
- Rotate by moving one track



Minesweeper: Outputs

- The rotation and velocity are adjusted by activating one or both of the left and right tracks

- The NN needs two outputs:
  - Left track
  - Right track

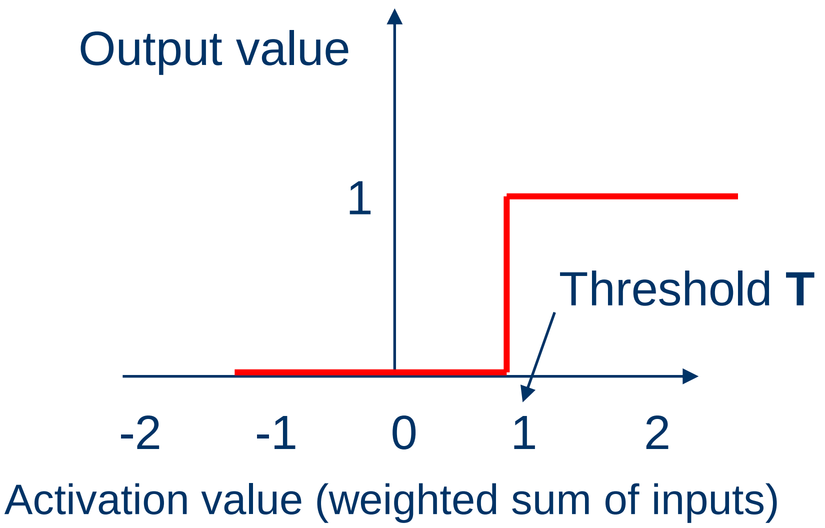


- Move forward by activating both tracks

Digression:

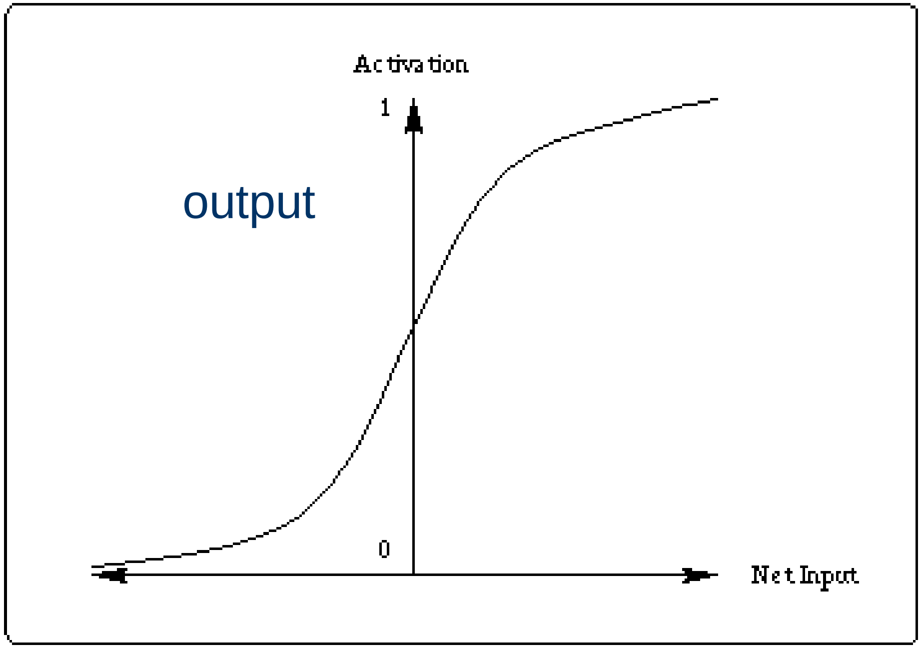
- When we talked about simple perceptrons we used a step function that output 0/1
- More useful to have a neuron that can output any value
  - Probability
  - Distance
  - Angle

Step Function:



Threshold Functions

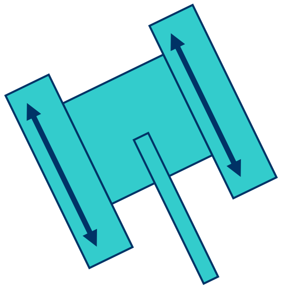
- Rather than a step activation function, it would be better to have one that varied:
  - Smoothly
  - Continuously
- Why ?
  - Can output a whole range of values
  - There is no abrupt change from one value to another
  - Imagine a car engine control system that allowed only 0 or maximum power



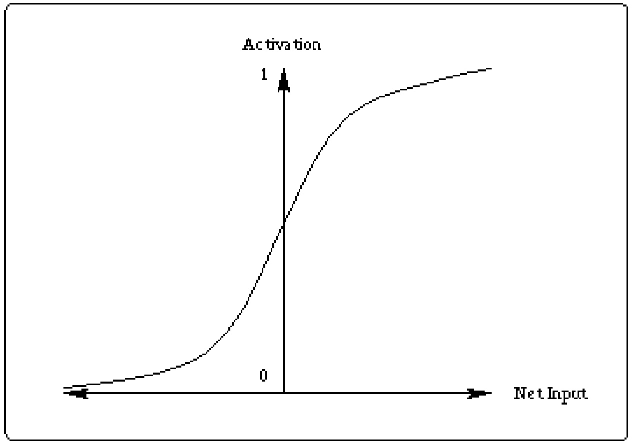
Input to neuron

Minesweeper: Outputs

- The rotation and velocity are adjusted by adjusting the **relative** speed of the left and right tracks



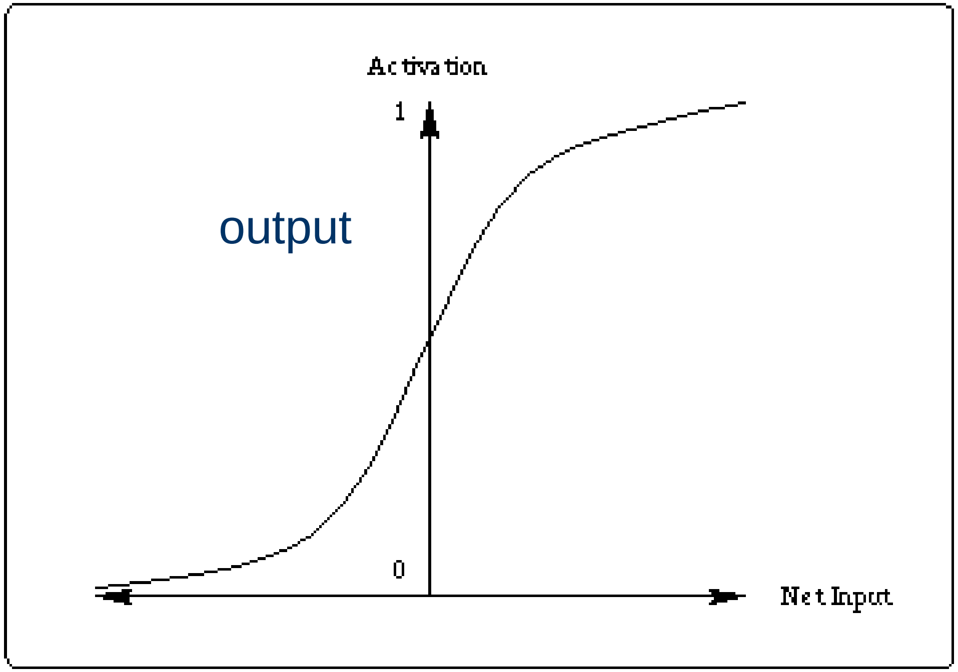
- The NN needs two outputs:
  - Left track speed
  - Right track speed
  - Smooth transition from 0 (off) to 1 (maximum)
  - Allows the robot to turn while moving position



Threshold Functions

- There are lots of mathematical functions that might work
- This one is called a **sigmoid function**
  - Varies between 0 and 1
  - Shape can be 'flattened' if required
  - It is define as:

$$\frac{1}{1+e^{-\frac{a}{p}}}$$



Input to neuron

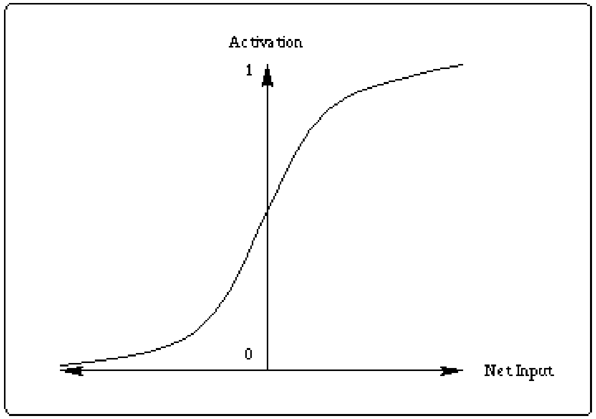
The Sigmoid Function

- a is the weighted sum

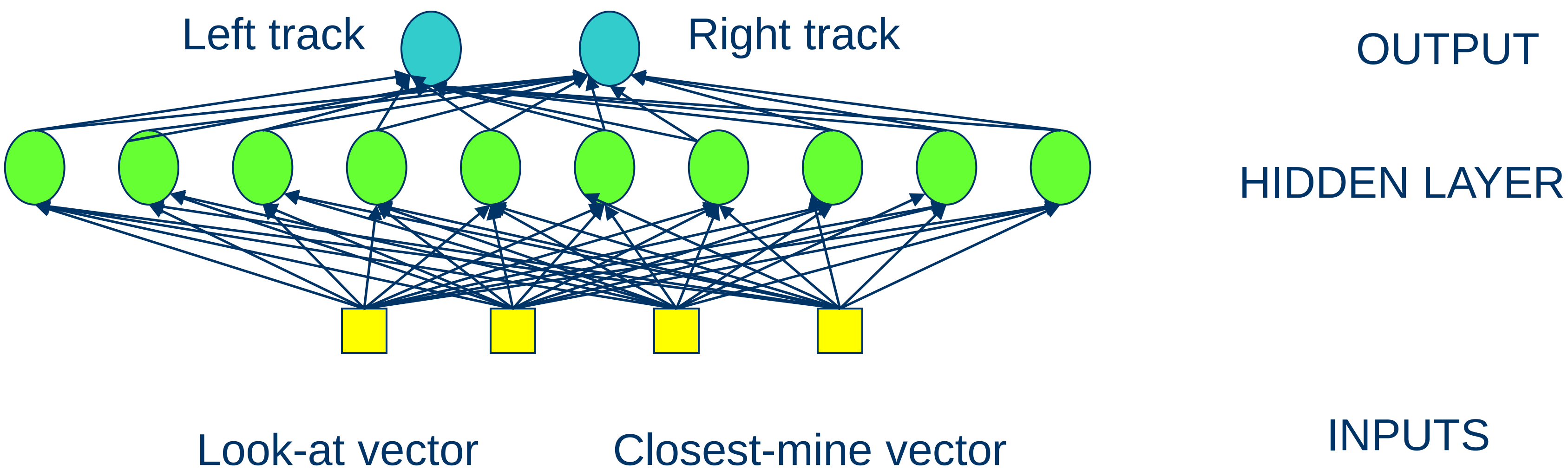
$$\sum w_i x_i$$

$$\frac{1}{1+e^{-\frac{a}{p}}}$$

- e is a mathematical constant 2.7183
- p is a user controlled parameter usually set to 1
- (by changing p we can ‘squash’ the curve)



The Minesweeper Neural Net



We need to assign each weight so that the network outputs the correct values for any set of possible inputs

Calculating the weights

- In this case, there are 72 weights to find
- For backpropagation, we need training data with input-output pairs:
  - Hard to obtain for this type of application:
- Another approach to training is to use an **Evolutionary Algorithm** to evolve the weights
  - A chromosome (length= number of weights) represents the weights in neural network

w_1	w_2	w_3	....	w_{71}	w_{72}

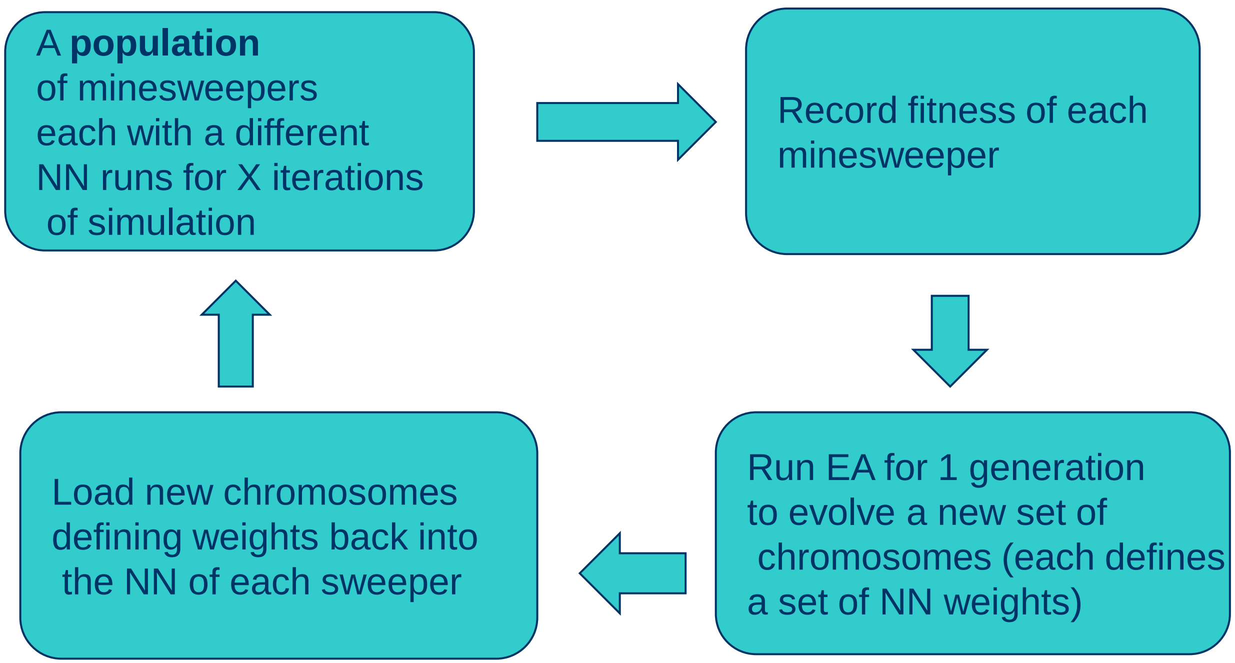
In-Game Training with an EA

- An EA needs a **representation**
  - Use floating point values, one for each weight we need
- and a **fitness function:**
  - Allow the game to run for some number of frames
  - Monitor how many mines each sweeper found

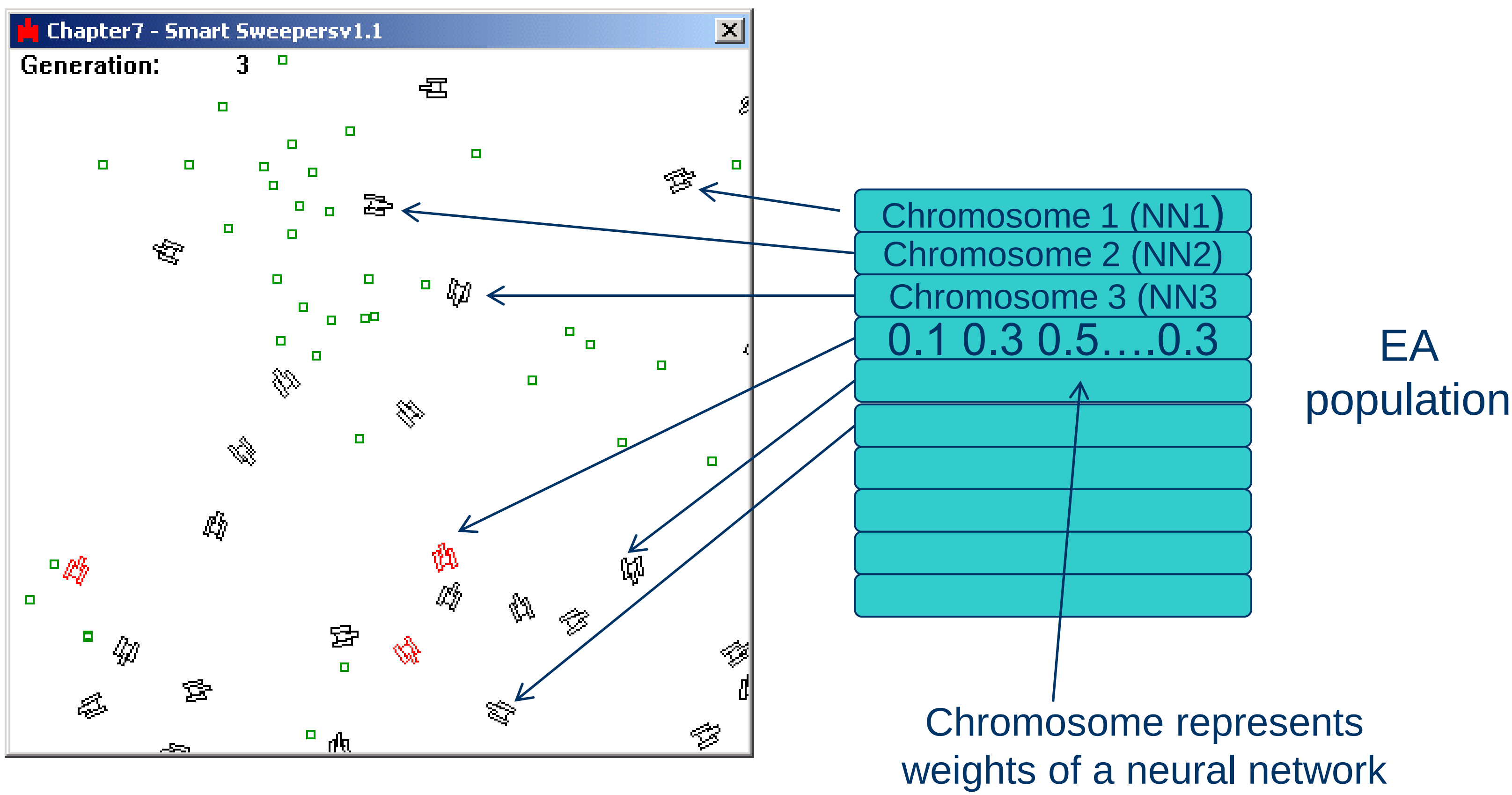
-0.1	0.3	0.6	0.4	……..	-0.8

Fitness = number of mines found in fixed time frame

The Flow of Control



Training with an EA



Flow of Control

Initialise a population of minesweepers each with its own neural network and its own chromosome representing the weights

Weights initially set to random values

While number of generations < maximum generations

- Run game (for a set amount of time)
- Record how many mines each sweeper detected
- Assign this value as the fitness of its chromosome
- Run the EA to evolve a new population of weights
  - Apply selection, crossover, mutation
- Insert new weights into each minesweepers NN

Repeat ….

Evolutionary Algorithm

- Generational EA (whole new generation produced each iteration)
- Differs from the steady state EA where typically 1 or 2 children are generated each generation

BEST Chromosome 1
Chromosome 2
Chromosome 3
.
.
.
.
Chromosome N

Generation

Copy best chromosome to new population **(elitism)**

While newPopSize < popSize
    Generate child
    - Selection
    - Crossover
    - Mutation
    Insert Child (**no replacement**)
Repeat

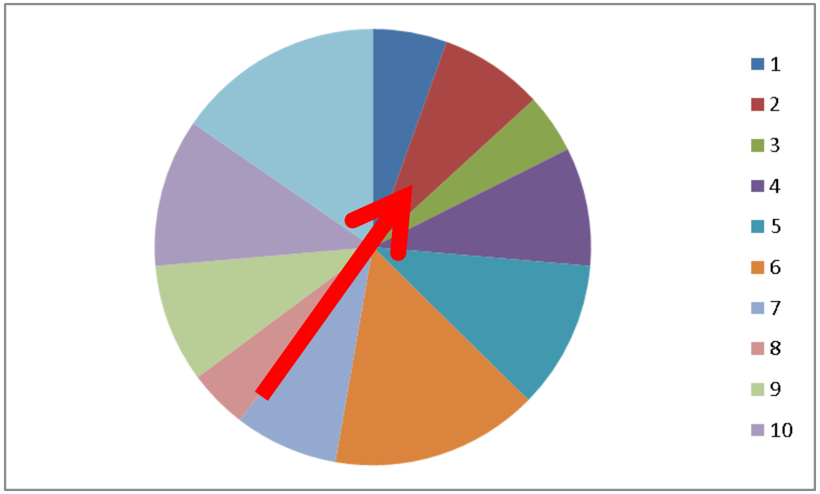
BEST Chromosome 1
Child Chromosome 2
Child Chromosome 3
.
.
.
.
Child Chromosome N

Evolutionary Algorithm

- In each generation:
  - Create new empty population
  - Copy best chromosome from current to new population (elitism)
  - Repeat until new population full:
    - Select two parents with **roulette wheel selection** from current population
    - Apply **crossover** to produce new child(ren)
    - Apply **mutation** to new child(ren)
    - Add child(ren) to new population
  - New population becomes current population

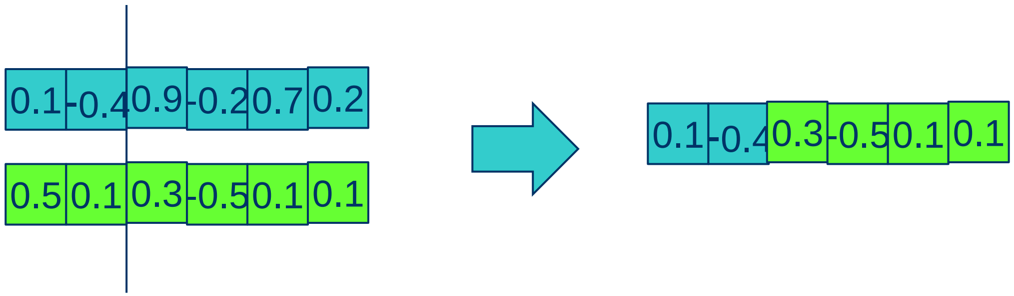
More details on the EA

- Selection:
  - Selection pressure is as a result of the selection stage only
  - No replacement stage in a generational EA
  - Best chromosome (could be more than 1. i.e. best 2) copied to new population each generation (elitism)
- Common to use Roulette Selection
  - Selection probability proportional to fitness



More details on the EA

- Crossover:
  - Any kind of crossover will produce legitimate chromosomes (1pt etc.)



  - Two point or uniform crossover will work just as well

More details on the EA

- Mutation
  - Adds or subtracts a small value from each gene

Probability of mutation

Random number between -1 and 1

```
for (int i=0;i<chromoSizel;i++){
   if (randFloat < mutationRate)
                 chromo[i] += randomClamped()*maxPerturbation;
}
```

Fix max size of change

Summary of EA+NN

Neural Network

- Inputs: 4
- Outputs: 2
- Hidden Layers: 1
- Hidden Neurons: 10

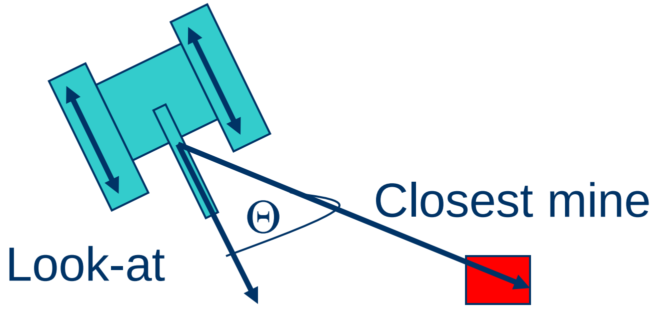
Evolutionary Algorithm

- Population Size: 30
- Selection Type: roulette
- Crossover type: 1 point
- Mutation Rate: 0.1
- Elitism: on
- Max Perturbation 0.3

Run for approx. 2000 generations to fully train networks

Some Improvements:

- We can reduce number of inputs even further
  - The important information is the **angle** between the two vectors
  - If we calculate the angle, we can just have one input to the network
  - **How ?**



Inputs

- We can calculate the angle using the **dot product**

  $\mathbf{A.B = |A|\ |B|\ \cos\theta}$

  $\mathbf{A.B = a_x b_x + a_y b_y}$

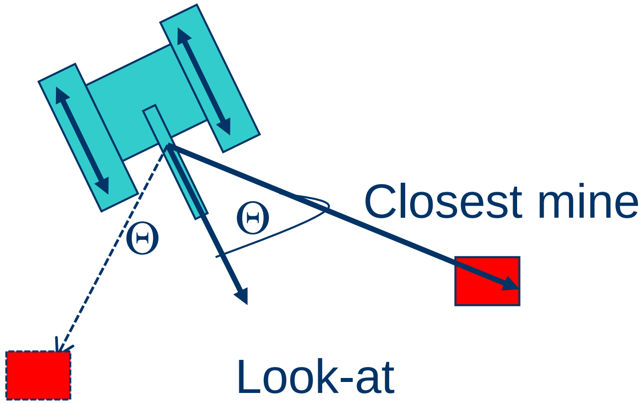
  $\mathbf{a_1 b_1 + a_2 b_2 = |A|\ |B|\ \cos\theta}$

  $\mathbf{a_1 b_1 + a_2 b_2 = \cos\theta}$ **(if magnitude normalised to 1)**

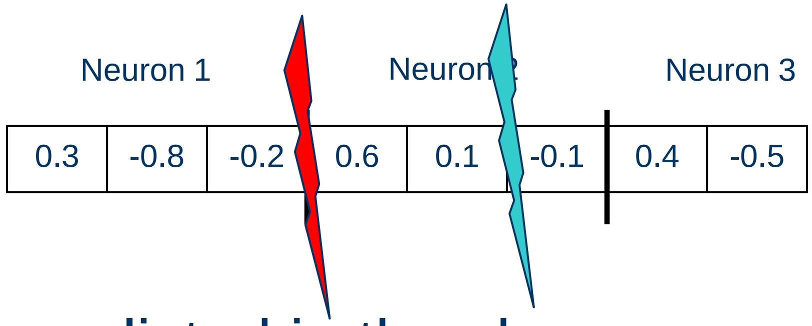
- We also need to know if the **relative position** of the mine to the heading (left or right): use a sign +/-, e.g.

  -30 = 30 degrees to left

  +15 = 15 degrees to right



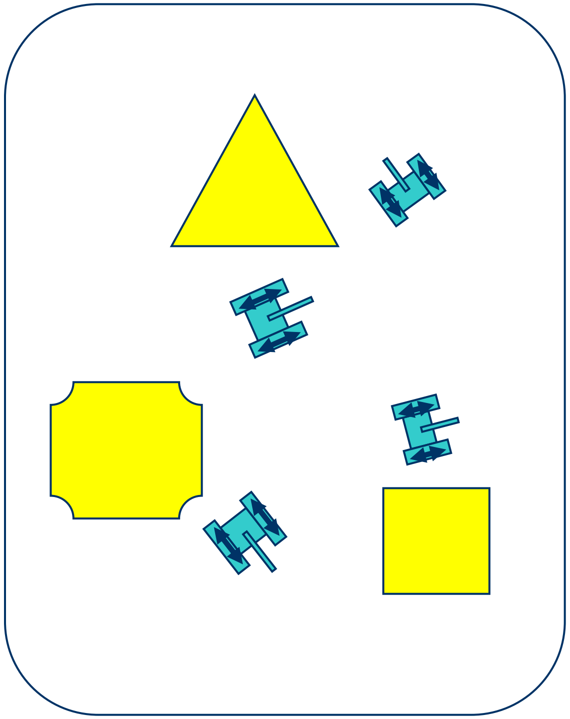
More improvements



- Weights are listed in the chromosome *per neuron*
- 1pt (or 2pt) crossover makes a random cut
- This can break up the weights for a single neuron
- Better to choose crossover points that only occur at the boundaries of neurons
  - So this limits the number of cut points

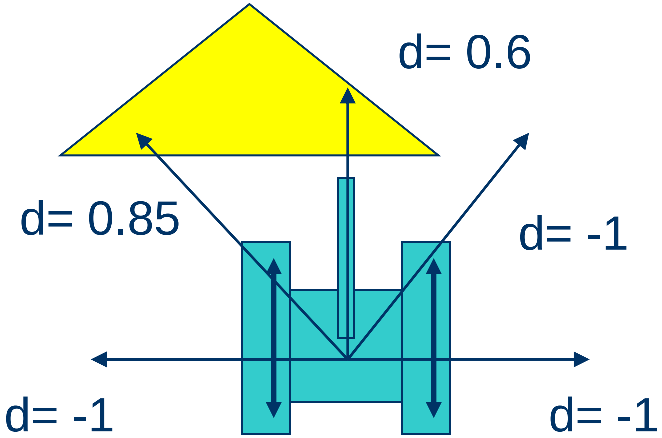
Obstacle Avoidance with NNs

- Obstacle avoidance common requirement in game AI
  - while still exploring the environment
- To do this successfully, agents need to:
  - perceive environment
  - take action to avoid collisions
- Typically implemented by adding sensors to the robot/car/agent etc….



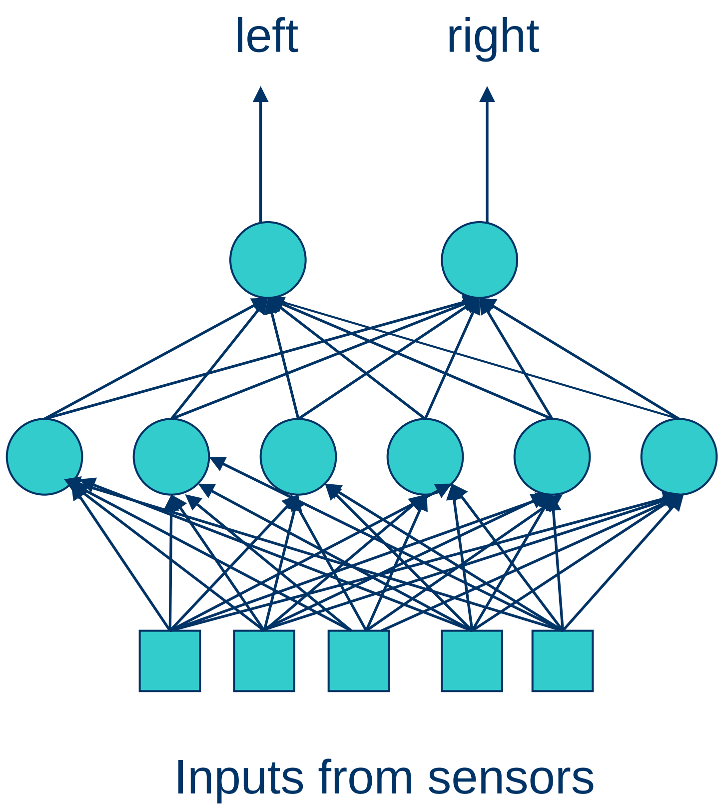
Obstacle Avoidance with NNs

- We can add sensors to each agent
- Adjust the number and range as required
- Each sensor returns:
  - -1 if there is no intersection between it and an obstacle
  - a value between 0 and 1 indicating the distance to the intersection otherwise
  - (closer to 0, closer to object)



Evolving an NN Controller

- Inputs to the NN are the readings from the 5 sensors
- Outputs again are the speeds of the left/right tracks
- Try one hidden layer at first
- Use an EA to evolve the correct weights



The Evolutionary Algorithm

- The EA needs a fitness function:
- What is a ‘good’ controller ?
  - Could record # collisions and penalise everytime it collides
    - Higher fitness = better minesweeper 
    - Could lead to negative scores 
- Better:
  - Record number of frames passed without colliding
    - More frames, higher fitness 
    - Doesn’t lead to negative scores 

The Fitness Function

- Does the fitness function look reasonable ?
  - Fitness: #frames passed without collision

```
if (!collided) fitness++;
```

The Fitness Function

- Does the fitness function look reasonable ?
  - Fitness: #frames passed without collision
- Robot will just rotate on the spot!!

```
if (!collided) fitness++;
```

The Fitness Function

- Does the fitness function look reasonable ?
  - Fitness: #frames passed without collision
- Robot will just rotate on the spot!!
- How can we fix this ?
  - Add extra reward for frames where there is zero (or little) rotation

```
if (!collided) fitness++;
if (rotation < rotationTolerance) fitness++;
```

Fitness Function

- Goal is to maximise fitness
- For each frame that passes:

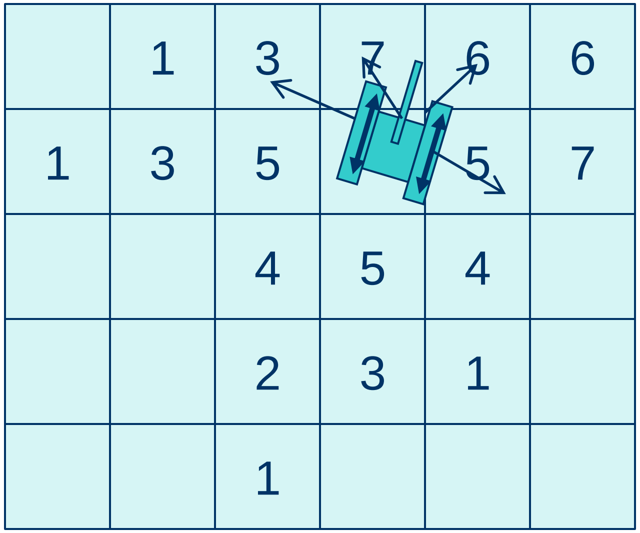
```
if (!Collided){
        fitness +=1
}

if (abs(Rotation) < RotationTolerance){
        fitness += 1
}
```

Amount robot rotated

Threshold defining max allowed rotation

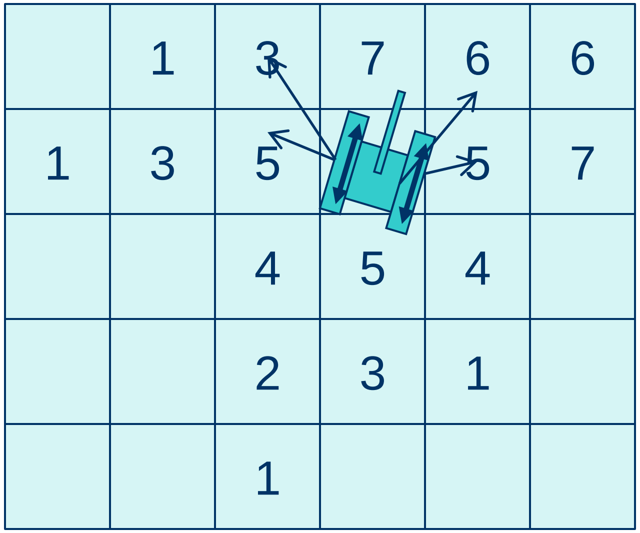
Further improvements

- We want the robot to explore the environment
- Add memory:
  - Record how many times each square is visited
  - Evolve networks that favour unvisited cells
- The sensors can 'feel' how many times a square has been visited



Further improvements

- Readings from sensors converted to scaled value between 0 and 1 representing times visited
- Sliding scale important to give robot a sense of ‘direction’



Fitness Function:

- We could combine the previous function with an extra factor:

  Fitness = NoRotation + NoCollision + Cells visited

- But....can just use number of cells visited:
  - Automatically will avoid obstacles
  - Automatically will stop spinning
  - (both slow them down and reduce fitness)

Uses in “real games”

- Calculate aiming for the AI characters in Quake ?
  - To prevent them being too accurate (…and therefore unrealistic)
- Network inputs:
  - Visibility (dark, foggy, bright)
  - Amount of target visible
  - Distance to target
  - Current weapon
- Output
  - Radius of distance from centre of target (bigger radius = less accurate)

Some additional thoughts

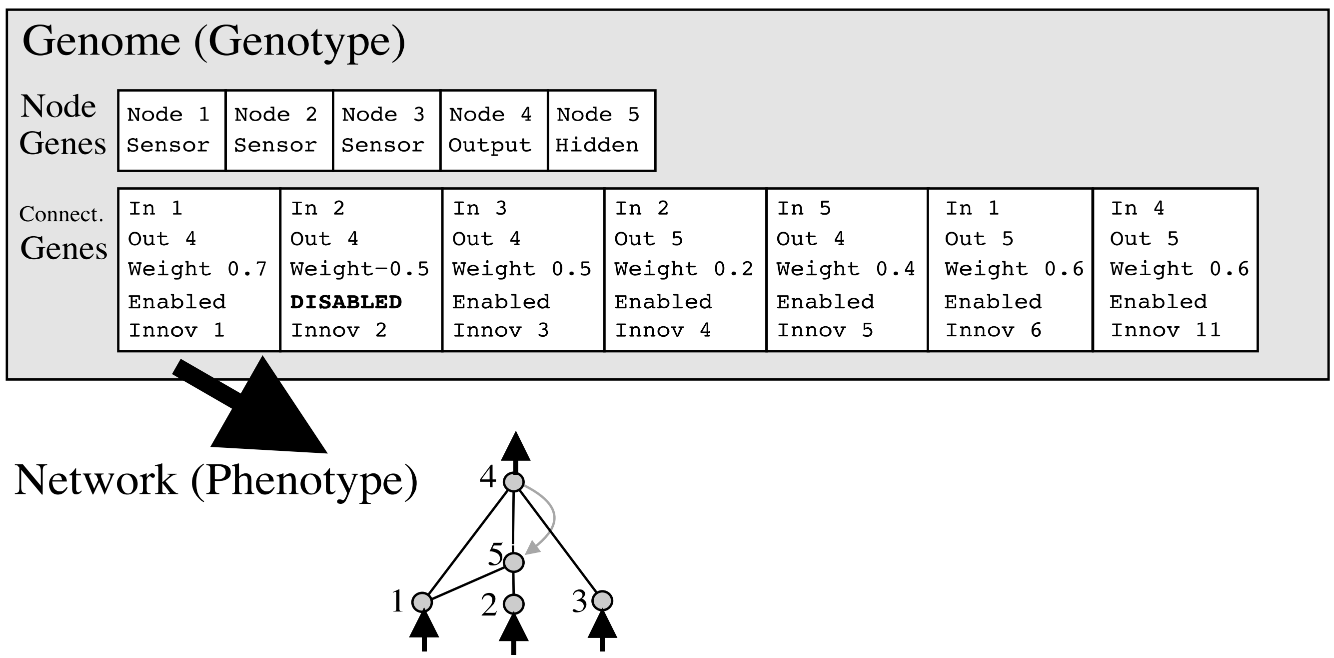
- In order to satisfy the fitness function many robot behaviours can emerge
  - Robot driving example
- You can also train a network to classify data using an EA
  - Fitness function = total error

Evolving Neural Network Topology

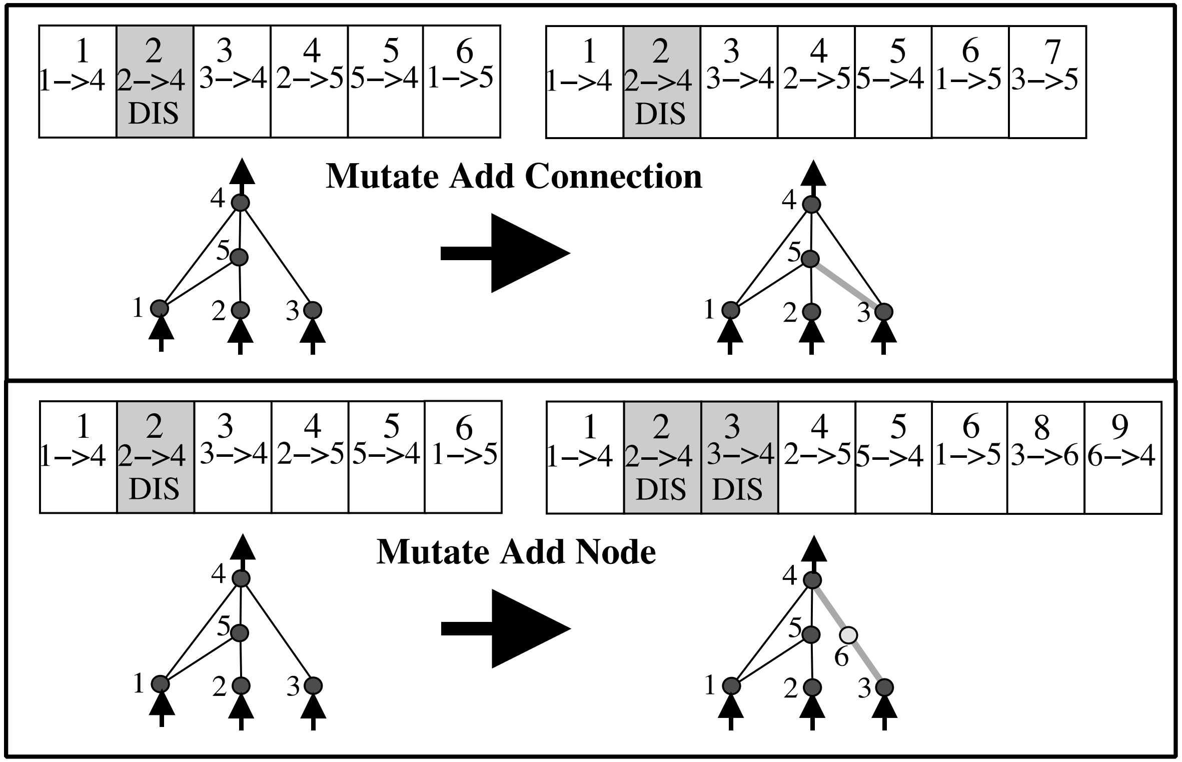
- As well as evolving weights, it makes sense to evolve topology as well
- A very well known technique is called NEAT:
  - Neuro-Evolution of Augmenting Topologies
  - Starts by using a population of networks with minimal technology that grow in complexity
- Generates neurons, connections, layers and weights

http://www.cs.ucf.edu/~kstanley/neat.html

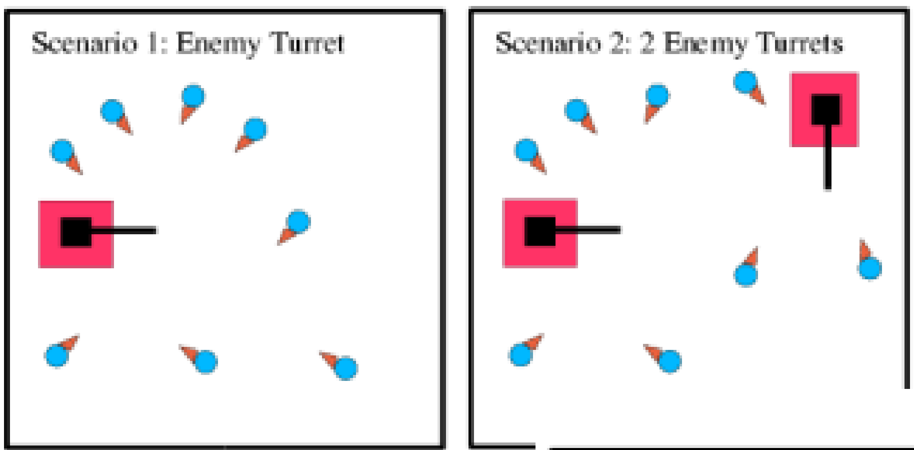
NEAT genomes

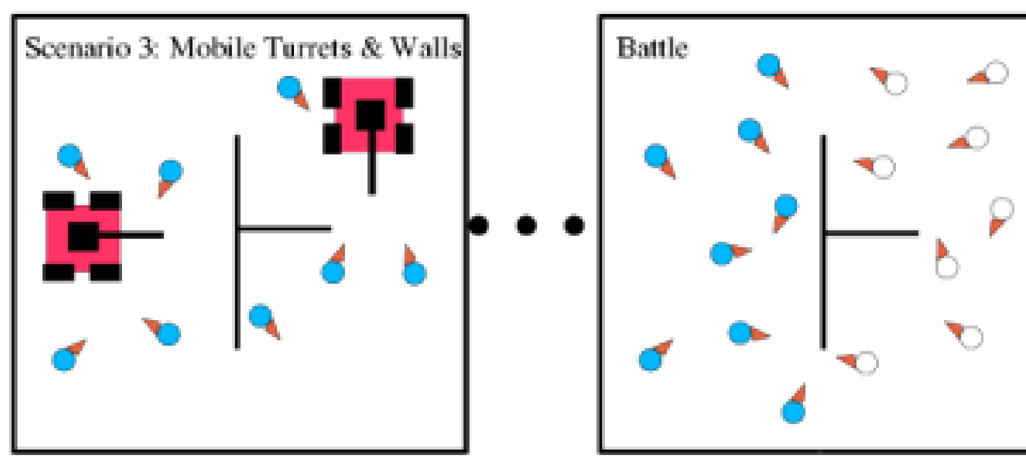


NEAT Mutation operators



Example: NERO





[Video]

[robot driving]

Comparison of Training Methods

Backpropagation

- Data needs to have input/output pairs
- Need plenty of data
- Careful data cleaning
- Training and testing required
- Lots of software packages available

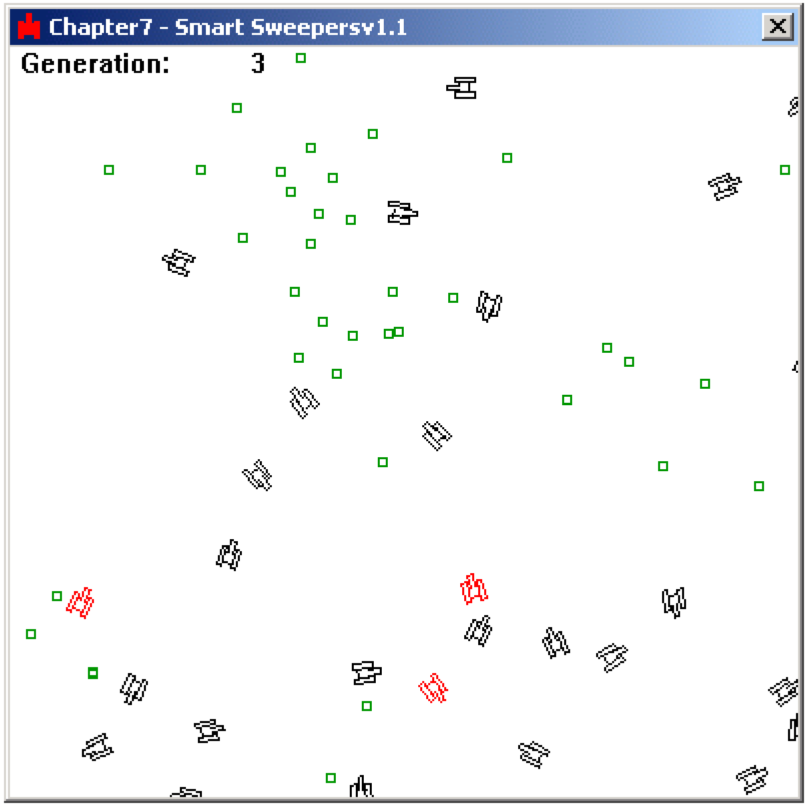
Evolutionary Algorithm

- Good when you don't have training data in the form of input/outputs pairs
- Careful design of fitness functions required
- Lots of parameters: need to deal with EA parameters as well as NN design
- Can be slow….

Practical

Evolving Minesweeper controller

- Source Code and Executable supplied with initialisation file
- Adapt topology and parameters by changing values in initialisation file
  - Conduct experiments to evaluate effect
- Implement own evolutionary operators by modifying C++ Code
  - Selection
  - Mutation
  - Crossover
- Other possibilities
  - Try different activation functions



Summary

- We have looked at a number of uses of Neural Networks:
  - Classification
  - Prediction
  - Control
- Training a network can be performed by backpropagation or using an evolutionary algorithm
  - Stochastic process in both cases
- A lot of effort goes into cleaning and preparing data
- Testing with unseen data very important