COMP250 Artificial Intelligence

10: Evolutionary Algorithms

Optimisation

Optimisation

Optimisation

- Define a **fitness function** $f(x)$

Optimisation

- Define a **fitness function** $f(x)$
- $f(x)$ **evaluates** a piece of content x, assigning it a **numerical score**

Optimisation

- Define a **fitness function** $f(x)$
- $f(x)$ **evaluates** a piece of content x, assigning it a **numerical score**
- **Higher** scores are **better**

Optimisation

- Define a **fitness function** $f(x)$
- $f(x)$ **evaluates** a piece of content x, assigning it a **numerical score**
- **Higher** scores are **better**
- We are exploring a **fitness landscape**

Running example

Running example

- https://github.com/Falmouth-Games-Academy/comp250-workshop-10

Running example

- `https://github.com/Falmouth-Games-Academy/comp250-workshop-10`
- Want to generate a map where there is a path from start to goal, and that path is as long as possible

Running example

- `https://github.com/Falmouth-Games-Academy/comp250-workshop-10`
- Want to generate a map where there is a path from start to goal, and that path is as long as possible
- Fitness measure:

$$f(x) = \begin{cases} \text{path length} & \text{if a path exists} \\ 0 & \text{otherwise} \end{cases}$$

Hillclimbing (a.k.a. gradient ascent)

Hillclimbing (a.k.a. gradient ascent)

- Start with an element x

Hillclimbing (a.k.a. gradient ascent)

- Start with an element x
- Create an element x' by making a **small change** to x

Hillclimbing (a.k.a. gradient ascent)

- Start with an element x
- Create an element x' by making a **small change** to x
  - May choose the small change at random

Hillclimbing (a.k.a. gradient ascent)

- Start with an element x
- Create an element x' by making a **small change** to x
  - May choose the small change at random
  - Or may try every possible change

Hillclimbing (a.k.a. gradient ascent)

- Start with an element x
- Create an element x' by making a **small change** to x
  - May choose the small change at random
  - Or may try every possible change
- If $f(x') > f(x)$, set $x = x'$

Hillclimbing (a.k.a. gradient ascent)

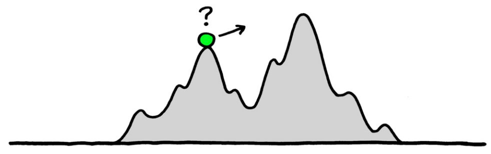
- Start with an element x
- Create an element x' by making a **small change** to x
  - May choose the small change at random
  - Or may try every possible change
- If $f(x') > f(x)$, set $x = x'$
- Otherwise, throw x' away and keep x as it is

Hillclimbing (a.k.a. gradient ascent)

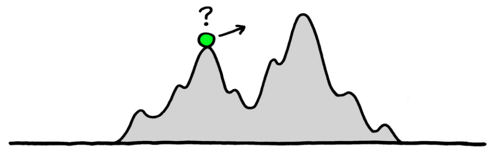
- Start with an element x
- Create an element x' by making a **small change** to x
  - May choose the small change at random
  - Or may try every possible change
- If $f(x') > f(x)$, set $x = x'$
- Otherwise, throw x' away and keep x as it is
- Repeat

Local optima

Local optima

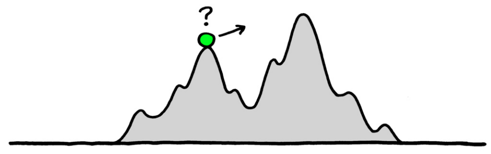


Local optima



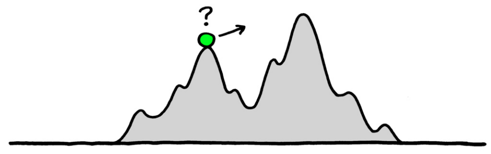
- Hillclimbing tends to get stuck at a **local optimum**

Local optima



- Hillclimbing tends to get stuck at a **local optimum**
- This may be much worse than the **global optimum**

Local optima



- Hillclimbing tends to get stuck at a **local optimum**
- This may be much worse than the **global optimum**
- Have to let the solution get worse before it gets better — hillclimbing doesn't allow this

Escaping the local optimum

Escaping the local optimum

- Shotgun search (a.k.a. random restart)

Escaping the local optimum

- Shotgun search (a.k.a. random restart)
  - Do several runs of hillclimbing from different starting positions

Escaping the local optimum

- Shotgun search (a.k.a. random restart)
  - Do several runs of hillclimbing from different starting positions
- Simulated annealing

Escaping the local optimum

- Shotgun search (a.k.a. random restart)
  - Do several runs of hillclimbing from different starting positions
- Simulated annealing
  - Probability of allowing the search to keep a worse solution

Escaping the local optimum

- Shotgun search (a.k.a. random restart)
  - Do several runs of hillclimbing from different starting positions
- Simulated annealing
  - Probability of allowing the search to keep a worse solution
  - This probability decreases as search progresses

FALMOUTH UNIVERSITY

Evolutionary algorithms

Evolutionary algorithms (EAs)

Evolutionary algorithms (EAs)

- Optimisation technique inspired by **biological evolution**

Evolutionary algorithms (EAs)

- Optimisation technique inspired by **biological evolution**
- We have a **population** of N solutions

Evolutionary algorithms (EAs)

- Optimisation technique inspired by **biological evolution**
- We have a **population** of N solutions
- Generation 0: choose N solutions at random

Evolutionary algorithms (EAs)

- Optimisation technique inspired by **biological evolution**
- We have a **population** of N solutions
- Generation 0: choose N solutions at random
- Generation $i + 1$: choose N new solutions based on the **fittest** individuals from generation i

Selecting the fittest

Selecting the fittest

- **All** individuals should have a **chance** of being selected

Selecting the fittest

- **All** individuals should have a **chance** of being selected
- But **fitter** individuals should be selected **more often**

Selecting the fittest

- **All** individuals should have a **chance** of being selected
- But **fitter** individuals should be selected **more often**
- Simple method: **tournament selection**

Selecting the fittest

- **All** individuals should have a **chance** of being selected
- But **fitter** individuals should be selected **more often**
- Simple method: **tournament selection**
  - Randomly choose t individuals

Selecting the fittest

- **All** individuals should have a **chance** of being selected
- But **fitter** individuals should be selected **more often**
- Simple method: **tournament selection**
  - Randomly choose t individuals
  - Select the fittest out of those t

Mutation

Mutation

- Select an individual

Mutation

- Select an individual
- Make a small change to it

Mutation

- Select an individual
- Make a small change to it
- Add the changed individual to the new population

Crossover

Crossover

- Select two individuals

Crossover

- Select two individuals
- Combine them somehow (take "half" of one and "half" of the other)

Crossover

- Select two individuals
- Combine them somehow (take “half” of one and “half” of the other)
- Add the resulting individual to the new population

Elitism

Elitism

- Take the top $x\%$ of generation i, and pass it straight through to generation $i + 1$

Not just for PCG

Not just for PCG

- Common use for optimisation: **parameter tuning**

Not just for PCG

- Common use for optimisation: **parameter tuning**
- Suppose we have several simple heuristic evaluation functions $h_1, h_2, \ldots, h_n$ which we want to combine into a single heuristic

Not just for PCG

- Common use for optimisation: **parameter tuning**
- Suppose we have several simple heuristic evaluation functions $h_1, h_2, \dots, h_n$ which we want to combine into a single heuristic
- **Linear combination**:

$$w_1 h_1 + w_2 h_2 + \cdots + w_n h_n$$

where $w_1, w_2, \dots, w_n$ are constants: **weights**

Not just for PCG

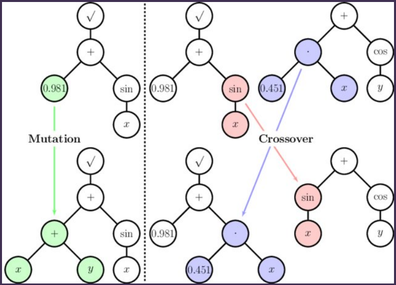
- Common use for optimisation: **parameter tuning**
- Suppose we have several simple heuristic evaluation functions $h_1, h_2, \ldots, h_n$ which we want to combine into a single heuristic
- **Linear combination**:

$$w_1 h_1 + w_2 h_2 + \cdots + w_n h_n$$

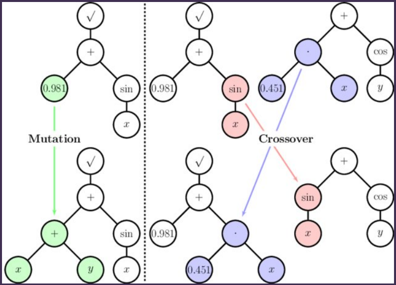
  where $w_1, w_2, \ldots, w_n$ are constants: **weights**
- What value to choose for the weights? This is an optimisation problem!

Genetic programming

Genetic programming

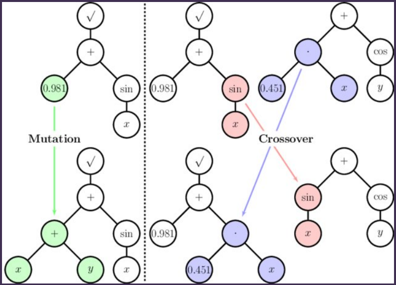


Genetic programming



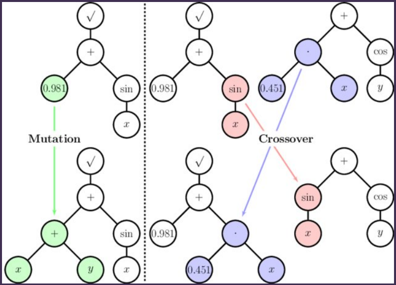
- Evolutionary algorithms for generating **code**

Genetic programming



- Evolutionary algorithms for generating **code**
- Typically uses a **tree-based** representation of code

Genetic programming



- Evolutionary algorithms for generating **code**
- Typically uses a **tree-based** representation of code
- Other approaches exist e.g. template-based

FALMOUTH
UNIVERSITY

Neural networks

Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs)

- **Inspired by** the structure of biological brains

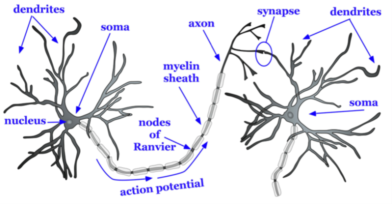
Artificial Neural Networks (ANNs)

- **Inspired by** the structure of biological brains
- Idea has been around since the 1950s

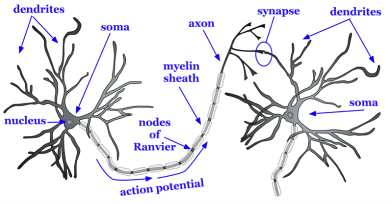
Artificial Neural Networks (ANNs)

- **Inspired by** the structure of biological brains
- Idea has been around since the 1950s
- Recent resurgence of interest: today's powerful CPUs and GPUs allow much larger ANNs to be used

Real neurons

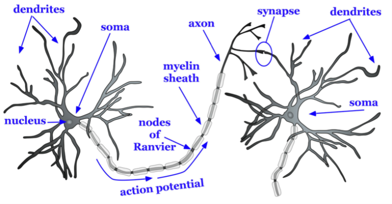


Real neurons



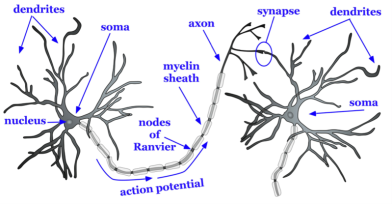
- An **electrically excitable** cell

Real neurons



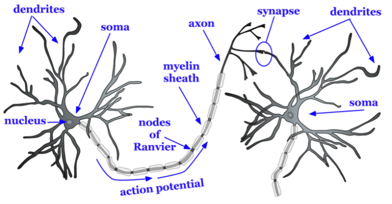
- An **electrically excitable** cell
- Neurons are **connected together**

Real neurons



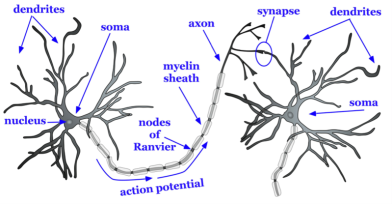
- An **electrically excitable** cell
- Neurons are **connected together**
- Connections can be **excitatory** or **inhibitory**

Real neurons



- An **electrically excitable** cell
- Neurons are **connected together**
- Connections can be **excitatory** or **inhibitory**
- If enough excitatory signals are received, the neuron **fires** — sends an electrical signal to the connected neurons

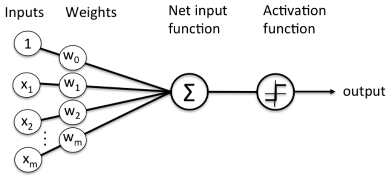
Real neurons



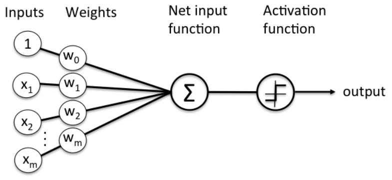
- An **electrically excitable** cell
- Neurons are **connected together**
- Connections can be **excitatory** or **inhibitory**

- If enough excitatory signals are received, the neuron **fires** — sends an electrical signal to the connected neurons
- Human brain contains approximately **100 billion** neurons

An artificial neuron

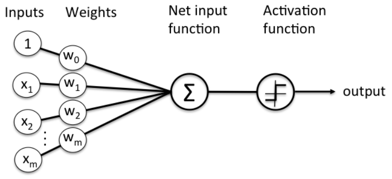


An artificial neuron



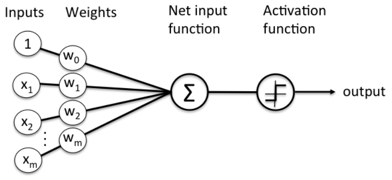
- A **perceptron**

An artificial neuron



- A **perceptron**
- Inputs $x_1, \ldots, x_m$ are outputs from **other perceptrons**

An artificial neuron



- A **perceptron**
- Inputs $x_1, \ldots, x_m$ are outputs from **other perceptrons**
- Each input has a **weight** w_i between -1 and $+1$

Perceptron activation

Perceptron activation

- The perceptron calculates a **weighted sum**

$$w_0 + w_1 x_1 + \cdots + w_m x_m$$

Perceptron activation

- The perceptron calculates a **weighted sum**

$$w_0 + w_1 x_1 + \cdots + w_m x_m$$

- This goes through an **activation function**

Perceptron activation

- The perceptron calculates a **weighted sum**

$$w_0 + w_1 x_1 + \cdots + w_m x_m$$

- This goes through an **activation function**
- Simplest: **step function**

$$\textbf{output} = \begin{cases} 1 & \text{if sum} \geq \text{threshold} \\ 0 & \text{if sum} < \text{threshold} \end{cases}$$

Perceptron activation

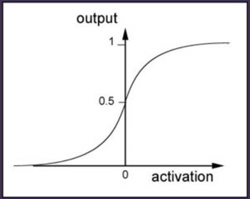
- The perceptron calculates a **weighted sum**

$$w_0 + w_1 x_1 + \cdots + w_m x_m$$

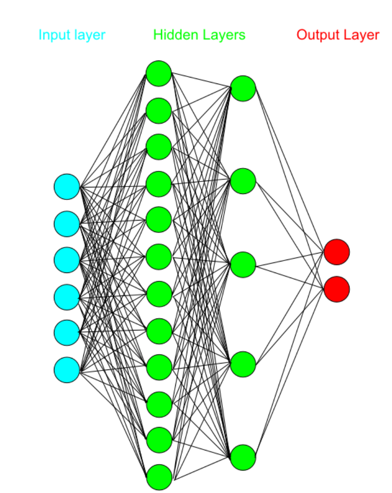
- This goes through an **activation function**
- Simplest: **step function**

$$\textbf{output} = \begin{cases} 1 & \text{if sum} \geq \text{threshold} \\ 0 & \text{if sum} < \text{threshold} \end{cases}$$

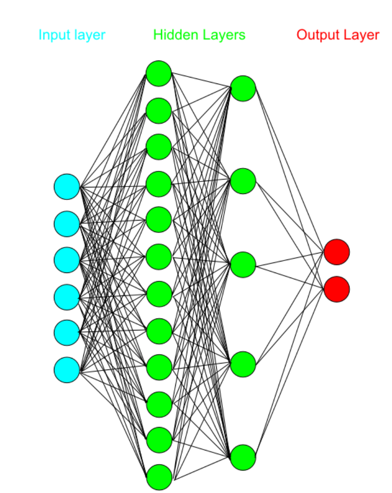
- More common: **sigmoid function**



An artificial neural network

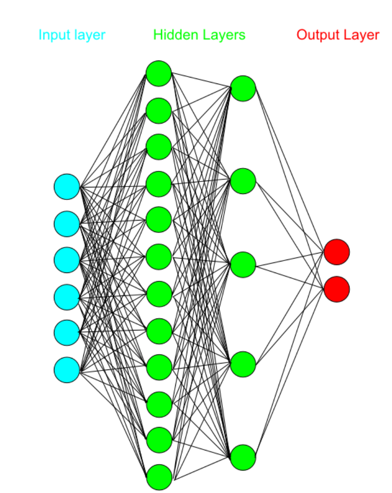


An artificial neural network



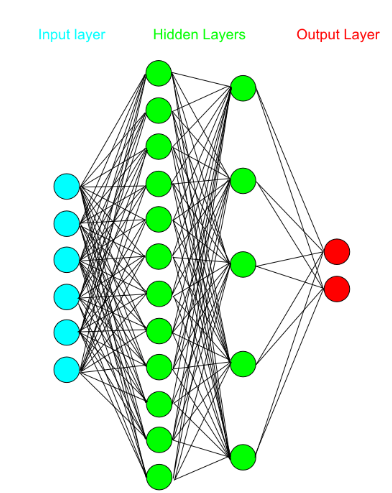
- A **multilayer perceptron (MLP)**

An artificial neural network



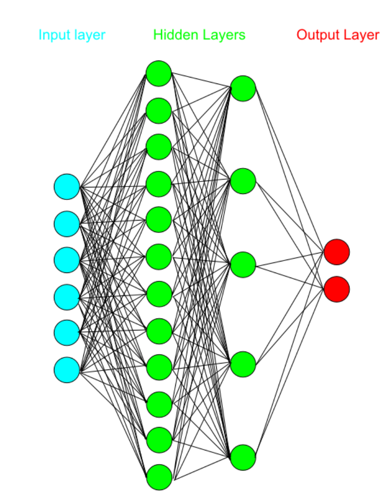
- A **multilayer perceptron (MLP)**
- Consists of an **input layer**, several **hidden layers** and an **output layer**

An artificial neural network



- A **multilayer perceptron (MLP)**
- Consists of an **input layer**, several **hidden layers** and an **output layer**
- Each layer is an array of **perceptrons**

An artificial neural network



- A **multilayer perceptron (MLP)**
- Consists of an **input layer**, several **hidden layers** and an **output layer**
- Each layer is an array of **perceptrons**
- Each perceptron's output is connected to **every** perceptron in the next layer

Image classification

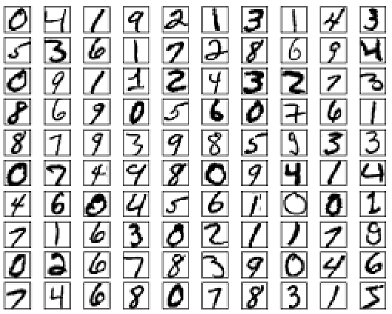
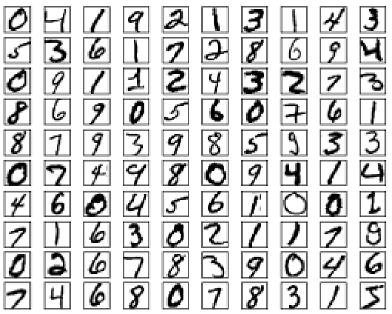
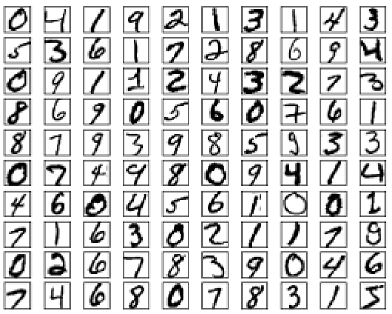


Image classification



- Classic example: **handwritten digit recognition**

Image classification



- Classic example: **handwritten digit recognition**
- Given a **raster image**, which of the digits 0 to 9 does it represent?

https://twitter.com/NaughtThought/status/846262063827730432

MLPs for image classification

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set
  - Output layer: 10 perceptrons

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set
  - Output layer: 10 perceptrons
- **Hidden layers**: ???

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set
  - Output layer: 10 perceptrons
- **Hidden layers**: ???
  - Parameters to tune

MLPs for image classification

- **Input**: pixels of the image, reduced down to 1 bit per pixel (i.e. black or white)
  - Input layer: 1 perceptron per pixel
- **Output**: 10 bits corresponding to digits 0 to 9, of which exactly one should be set
  - Output layer: 10 perceptrons
- **Hidden layers**: ???
  - Parameters to tune
- **Weights**: ???

How to set the weights?

How to set the weights?

- We need to **train** the network

How to set the weights?

- We need to **train** the network
- Idea:

How to set the weights?

- We need to **train** the network
- Idea:
  - Feed in **training data**

How to set the weights?

- We need to **train** the network
- Idea:
  - Feed in **training data**
  - When the network happens to give the correct answer, **reinforce** the relevant weights

How to set the weights?

- We need to **train** the network
- Idea:
  - Feed in **training data**
  - When the network happens to give the correct answer, **reinforce** the relevant weights
  - Repeat until a desired **accuracy** is obtained

How to set the weights?

- We need to **train** the network
- Idea:
  - Feed in **training data**
  - When the network happens to give the correct answer, **reinforce** the relevant weights
  - Repeat until a desired **accuracy** is obtained
- Note: this requires a large amount of training data that is **tagged**, i.e. for which we already know the correct answer

Stochastic gradient descent

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**
- Want to minimise the **error** over the training data

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**
- Want to minimise the **error** over the training data
- **Stochastic**: perform several training **epochs**

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**
- Want to minimise the **error** over the training data
- **Stochastic**: perform several training **epochs**
- Each epoch uses a randomly sampled **subset** of the training data

Stochastic gradient descent

- **Gradient descent**: opposite of **gradient ascent** a.k.a. **hillclimbing**
- Want to minimise the **error** over the training data
- **Stochastic**: perform several training **epochs**
- Each epoch uses a randomly sampled **subset** of the training data
- This reduces computation time, and helps to escape local optima

ANN example

`http://playground.tensorflow.org`

Overfitting

Overfitting

- ANN learns **patterns** in the training data

Overfitting

- ANN learns **patterns** in the training data
- Insufficient training data might result in the network learning "patterns" that are actually random anomalies

Next time

Next time

- Deep learning

Next time

- Deep learning
- Basically, large and computationally intensive neural networks

Next time

- Deep learning
- Basically, large and computationally intensive neural networks
- Currently state-of-the-art in AI

Workshop — more MicroRTS